Text File IO

Class 17

Data

- variables are storage locations for data in RAM
- RAM is volatile
- its contents vanish when the program ends

- to make data persist across different runs of a program
- and across different programs
- we store data in files on disk

Files

- a file on disk is strictly a sequence of bytes
- when you ask the operating system for some stuff from a file, you just get raw bytes
- it is up to you, the programmer, how to interpret those bytes

Files

- a file on disk is strictly a sequence of bytes
- when you ask the operating system for some stuff from a file, you just get raw bytes
- it is up to you, the programmer, how to interpret those bytes
- there are two main flavors of file
  1. binary files
  2. text files
- every file is a binary file in the sense that it contains bytes
- text files, however, contain only bytes that correspond to ASCII characters
- one of those bytes represents the newline character, interpreted as the end of a line
- thus text files are easy to interpret as a sequence of lines each of which is a sequence of characters

Values

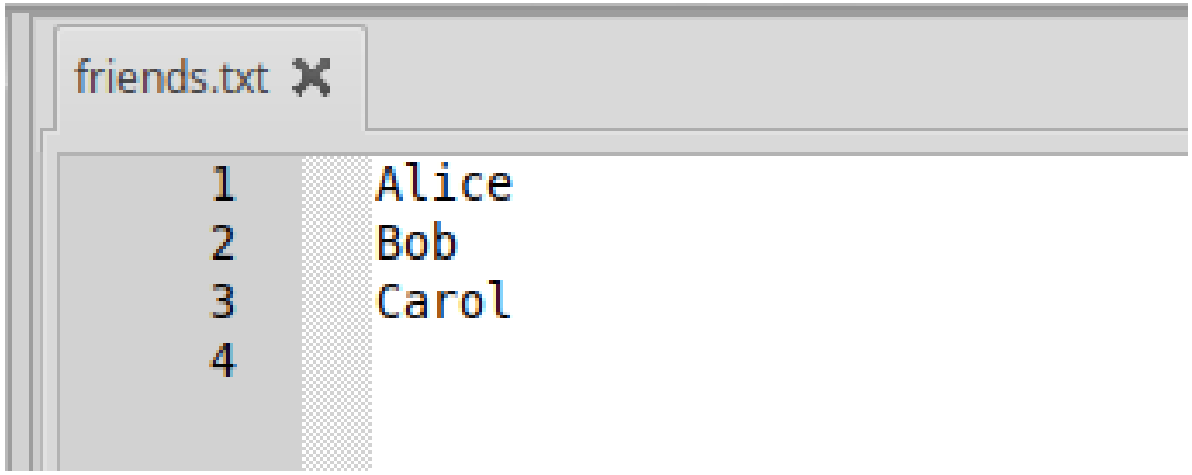
- a digital computer can only manipulate bits, 0s and 1s
- a group of 8 bits is a byte
- a byte can be interpreted as an unsigned integer value in the range 0 – 255
- thus, the only fundamental values a computer can manipulate are the 256 values in the range 0 – 255
- at the lowest level, there are no characters, or doubles, or negative numbers, or strings
- there are only the binary values 0 – 255

Characters

- to represent a character, a program must use an encoding
- an encoding is an agreement about which bit pattern will represent which character
- e.g., let us agree that in a character context the byte 0100 0001 (which is 65_{10} or `0x41`) will represent 'A'

- by default, C++ uses the ASCII encoding scheme, which we have seen several times

Text Files



what we see in an editor

A	l	i	c	e	\n	B	o	b	\n	C	a	r	o	l	\n

what is really in the disk file

- this is why every line of output must be terminated with a newline

Binary Files

- all files contain bytes
- the bytes encode some information, that we call data
- binary files contain data that is strictly designed to be read by computer programs
- some are open standard formats, e.g., jpeg and pdf for images
- some are proprietary, e.g., xls for spreadsheets and dwg for CAD drawings
- this is not the same as structured text files that conform to a format standard such as csv or xml

Text Files

- text files contain data that can be read either by a human or by a computer program
- working with text files requires the program to be able to

  read copy the data from the disk file into a program's variables

  write copy data from a program's variables out to some space on disk

Streams

- the model that C++ uses for working with files is to consider them as streams of bytes
- to read from a file is to treat the file as an input stream of bytes coming in from disk
- to write to a file is to treat the file as a destination for an output stream of bytes going out to disk

Streams

- the model that C++ uses for working with files is to consider them as streams of bytes
- to read from a file is to treat the file as an input stream of bytes coming in from disk
- to write to a file is to treat the file as a destination for an output stream of bytes going out to disk

- fortunately, we already know how to deal with streams
- for input, we use the stream extraction operator $\gg$ and the function getline
- for output, we use the stream insertion operator $\ll$

Filenames

- files on disk are identified by filename
- by convention a filename consists of name and extension
- e.g., `grades.xls` or `phone_plan.cpp`
- by default, Windows file explorer does not show you the extension, because Windows was made for your grandma, not for programmers
- you should always turn on the listing of full file details so you can see file details, including the extension
- the extension indicates what kind of file it is
- there are hundreds of extensions
- we will use .txt for plain text files
- we will always assume that the text file is in the current directory (the project directory of CodeBlocks), so we will not have to worry about paths which are different on different operating systems

Opening an Input File

- to access a disk file it must first be opened
- to open a file means to associate it with a special file variable
- the file variable must be of type ifstream for an input text file
- this type is defined in the `<fstream>` library

```
#include <fstream>
...
ifstream input_file;
input_file.open("foo.bar");
```

- the identifier input_file is a programmer-defined name for internal use in the program
- the string "foo.bar" represents the external name of the file on disk

Opening an Output File

```
#include <fstream>
...
ofstream output_file;
output_file.open("foo.bar");
```

- the file variable must be of type ofstream for an output text file
- if the file does not already exist, it is created in the current directory

Opening an Output File

```
#include <fstream>
...
ofstream output_file;
output_file.open("foo.bar");
```

- the file variable must be of type ofstream for an output text file
- if the file does not already exist, it is created in the current directory
- WARNING! if the file does already exist, all its contents are deleted by the open function call

Closing a File

- before your program terminates
- you must close the file
- this frees up the operating system resources associated with the file
- for output files especially, this flushes the write buffer and ensures that everything you tried to write to the file is actually stored on disk

```
output_file.close();
```

Programs That Write to Files

- a newly opened file is empty
- each successive output operation appends more data after that already written
- cannot back up
- see files
  - Program 5-15 (page 274): write a few strings
  - Program 5-16: write strings without newlines (bad)
  - Program 5-17: numeric values entered from the keyboard, written to disk file
  - Program 5-18: string values entered from the keyboard, written to disk file

Programs That Read from Files

- the open function places the read marker at the first byte of the file
- as data are extracted from the file, the read marker is advanced toward the end of the file
- cannot back up

Programs That Read from Files

- the open function places the read marker at the first byte of the file
- as data are extracted from the file, the read marker is advanced toward the end of the file
- cannot back up

- if data are numeric, use stream extraction
  `input_file >> value;`
- if data are strings, use getline to read an entire line
  `getline (input_file, a_string);`
- see program 5-19, which uses getline instead of $\gg$ because it's reading strings

End of File

- a file may contain a little data
- or a lot
- typically we don't know how many lines of data a file contains
- the stream extraction operator $\gg$ returns a value
- it returns true if the read was successful
- it returns false if the read was not successful
- this allows us to control a while loop with the extraction operator

- see program 5-22

File Open Errors

- there are various conditions that may cause an open function to fail
  - attempt to read a file that does not exist
  - attempt to write (create) a file without OS permissions in that directory
  - attempt to write (create) a file on a disk that's full

File Open Errors

- there are various conditions that may cause an open function to fail
  - attempt to read a file that does not exist
  - attempt to write (create) a file without OS permissions in that directory
  - attempt to write (create) a file on a disk that's full

- in general, always need to check whether the open function succeeded
- do not use the method of program 5-23
- instead use method of middle of page 286 and program 5-24:

```
input_file.open(filename);
if (!input_file.fail())
{
   ...
```

Read and Write

- it is possible to open a file for both reading and writing at the same time
- but it is complex and uncommon
- for now, we will only allow reading or writing on any given file but not both

Read and Write

- it is possible to open a file for both reading and writing at the same time
- but it is complex and uncommon
- for now, we will only allow reading or writing on any given file but not both

- but it is completely acceptable to have two files open at the same time
  - one for reading
  - one for writing

A More Realistic Example

- the examples shown so far have been relatively simplistic
- as a more realistic example, consider a file structured as shown below
- for use as input to a program to generate phone plan invoices

```
A Rachel Carson
12.3
C Aldo Leopold
A Paul R. Erlich
7.25
B Farley Mowat
1.23
```

Omitted

- we are using C++11
- therefore the top half of page 288 is irrelevant
- expunge those nasty old C-strings whenever you can!

getline and Stream Extraction

- you are familiar with stream extraction $\gg$ that skips whitespace and stops at either
  - whitespace
  - a character not valid for the variable being read (e.g., a decimal point for an integer)

getline and Stream Extraction

- you are familiar with stream extraction $\gg$ that skips whitespace and stops at either
  - whitespace
  - a character not valid for the variable being read (e.g., a decimal point for an integer)

- you are familiar with getline that reads a string from the current position to the newline, skips the newline, and positions the read marker at the beginning of the next line

getline and Stream Extraction

- you are familiar with stream extraction $\gg$ that skips whitespace and stops at either
  - whitespace
  - a character not valid for the variable being read (e.g., a decimal point for an integer)

- you are familiar with getline that reads a string from the current position to the newline, skips the newline, and positions the read marker at the beginning of the next line

- there is no problem with a getline followed by a stream extraction
- but stream extraction followed by a getline can cause major problems

Data Files

- consider a data file that has alternating lines of data
- the first line consists of name
- the second line has numbers

B	o	b	\n	
4	7	2	9	\n
A	n	n	e	\n
1	2	3	\n	

Getline and Stream Extraction

- we use getline for the first name
- no problem, the read marker reads B-o-b
- skips *newline*
- finishes on the 4

B	o	b	\n	
4	7	2	9	\n
A	n	n	e	\n
1	2	3	\n	

Getline and Stream Extraction

- we now use stream extraction $\gg$ for the number
- no problem, we read 4-7-2-9 and the read marker ends on the *newline*

B	o	b	\n	
4	7	2	9	\n
A	n	n	e	\n
1	2	3	\n	

Getline and Stream Extraction

- now we want to use getline again to read Anne
- but the read marker is on the newline at the end of 4729
- getline reads until the next newline but it's already on a newline
- so it reads 0 characters, skips the newline, and puts the read marker on the A of Anne

B	o	b	\n	
4	7	2	9	\n
A	n	n	e	\n
1	2	3	\n	

Getline and Stream Extraction

- now everything is hosed
- we're ready to read an integer
- but the read marker is on a character

B	o	b	\n	
4	7	2	9	\n
A	n	n	e	\n
1	2	3	\n	

Getline and Stream Extraction

- solution: after extraction $\gg$, we have to skip over the newline
- to read up to and skip over the newline, we use getline with a dummy variable: `getline(stream, dummy);`
- in programming a variable named dummy is used for a value that is never used
- a variable named `dummy` signals that its value will be ignored

```
getline(stream, string_variable);
stream >> variable >> variable;
getline(stream, dummy); // to consume the newline
getline(stream, string_variable);
stream >> variable >> variable;
getline(stream, dummy); // to consume the newline
...
```