And Even More and More C++

Outline

- C++ Classes
  - Friendship
  - Inheritance
  - Multiple Inheritance
  - Polymorphism
  - Virtual Members
  - Abstract Base Classes
- File Input/Output

File Input/Output

- We've already done keyboard input and screen output
  - But that doesn't preserve data
    - The Homework.cpp lab assignment is fairly useless since all the entered data goes away when you quit the program
- Just like in Python, we can read from files and write to files
  - **ofstream**: stream used to write to file (output file stream)
  - **ifstream**: stream used to read from a file (input file stream)
  - **fstream**: stream to both read from and write to a file (file stream)

File Input/Output Example

```
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile;
  myfile.open ("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

```
[file example.txt]
Writing this to a file.
```

- Declare a file by its operation type
- Open the file
- Perform read/write operations on the file
- Close the file

Opening a File

- A file can be opened in different modes:

ios::in	Open for input operations.
ios::out	Open for output operations.
ios::binary	Open in binary mode.
ios::ate	Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
ios::app	All output operations are performed at the end of the file, appending the content to the current content of the file.
ios::trunc	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.

- Modes can be combined using the bitwise or operator (|):

```
ofstream myfile;
myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

class	default mode parameter
ofstream	ios::out
ifstream	ios::in
fstream	ios::in | ios::out

Files

- Can have either text or binary files
- Can create and open file in a single statement:

```
ofstream myfile ("example.bin", ios::out | ios::app | ios::binary);
```

- Can then check to see if the file opened successfully:

```
if (myfile.is_open()) { /* ok, proceed with output */ }
```

- After file input/output is complete, should close the file:

```
myfile.close();
```

Moving Around in a File

- All file streams keep track of at least one position in the file
- get and put positions
  - Input streams keep an internal “get” position
    - This is where the next data item will be read from
  - Output streams keep an internal “put” position
    - This is where the next data item will be written
- tellg() and tellp() are functions that retrieve the position
- seekg() and seekp() allow you to move the position

Moving Around in a File

- Can use absolute positioning or relative positioning
  - seekg(position) is absolute – it will go to that position
  - seekg(offset, direction) is relative – it will move "offset" number of positions past "direction"
    - direction can be:

ios::beg	offset counted from the beginning of the stream
ios::cur	offset counted from the current position
ios::end	offset counted from the end of the stream

Example

```
// obtaining file size
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  streampos begin,end;
  ifstream myfile ("example.bin", ios::binary);
  begin = myfile.tellg();
  myfile.seekg (0, ios::end);
  end = myfile.tellg();
  myfile.close();
  cout << "size is: " << (end-begin) << " bytes.\n";
  return 0;
}
```

```
size is: 40 bytes.
```

Binary Files

- We use the >> and << operators to write to text files
- This is not efficient for binary files
  - Use read and write instead

```
// reading an entire binary file
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  streampos size;
  char * memblock;

  ifstream file ("example.bin", ios::in|ios::binary|ios::ate);
  if (file.is_open())
  {
    size = file.tellg();
    memblock = new char [size];
    file.seekg (0, ios::beg);
    file.read (memblock, size);
    file.close();

    cout << "the entire file content is in memory";

    delete[] memblock;
  }
  else cout << "Unable to open file";
  return 0;
}
```

```
the entire file content is in memory
```

Buffers and Synchronization

- When we work with files, they are associated with an internal buffer of type streambuf
  - This is so that we don't write to disk for every single piece of data
- When the buffer is flushed, that is when the data is actually written to disk
  - Called synchronization
  - Happens when:
    - File is closed by the program
    - When the buffer is full
    - Explicitly (you can force a buffer flush with the flush() command)
    - Explicitly with sync()

Summary

- C++ Classes
  - Friendship
  - Inheritance
  - Multiple Inheritance
  - Polymorphism
  - Virtual Members
  - Abstract Base Classes
- File Input/Output