Basics of File I/O and Stream Objects

Input/Ouput to and from files

- File input and file output is an essential in programming.
    - Most software involves more than keyboard input and screen user interfaces.
    - Data needs to be stored somewhere when a program is not running, and that means writing data to disk.
    - For this, we need file input and file output techniques.
- Fortunately, this is **EASY** in C++!
    - If you know how to do screen output with `cout`, and keyboard input with `cin`, then you already know most of it!
    - File I/O with streams works the same way. The primary difference is that objects other than `cout` and `cin` will be used

Kinds of Files

- **Formatted Text** vs. **Binary** files
    - A *text* file is simply made of readable text characters. It looks like the output that is typically printed to the screen through the `cout` object
    - A *binary* file contains unformatted data, saved in its raw memory format. (For example, the integer 123456789 is saved as a 4-byte chunk of data, the same as it's stored in memory - NOT as the 9 digits in this sentence).
- **Sequential** vs. **Random Access** files
    - A *sequential* file is one that is typically written or read from start to finish
    - A *random access* file is one that stores records, all of the same size, and can read or write single records in place, without affecting the rest of the file
- (For now, we'll use sequential text files)

Creating file stream objects, and attaching to files

- `cout` and `cin` are **objects**
    - `cout` is the standard output stream, usually representing the monitor(i.e., the screen). It is of type `ostream`
    - `cin` is the standard input stream, usually representing the keyboard. It is of type `istream`
    - `ostream` and `istream` are **classes**
    - If you were to have declared them, you might have written:

```
ostream cout;
istream cin;
```

- To create file stream objects, we need to include the `<fstream>` library:
- `#include <fstream>`

- `using namespace std;`
- This library has classes **`ofstream`** ("output file stream") and **`ifstream`** ("input file stream"). Use these to declare file stream objects:

```
ofstream out1; //create a file output stream, called out1
ifstream in1;  // create a file input stream, called in1
```

- File stream objects need to be attached to files before they can be used. Do this with a *member function* called **`open`**, which takes in the filename as an argument:

```
out1.open("outfile1.txt"); // for ofstreams, this call creates

in1.open("infile1.txt");  //for ifstreams, this call tries to open
                          //and read
```

- Will `open()` always work?
    - Since it's possible for `open()` to fail, one should always check to make sure there's a valid file attached
    - One easy way is to test the true/false value of the stream object itself. A stream that is not attached to a valid file will evaluate to "false"

```
if (!in1)// if in1 not attached to a valid input source, abort
{
     cout << "Sorry, bad file.";
     exit(0);       // special system call to abort program
                    //  may require <cstdlib> to be included
}
```

- When finished with a file, it can be detached from the stream object with the member function `close()`:

```
in1.close();
out1.close();
```

Note that the `close` function simply closes the file. It does not get rid of the stream object. The stream object can now be used to attach to another file, if desired:

```
in1.open("infile2.txt");
```

Using file streams

- Once a file stream object is attached to a file, it can be used with the same syntax as `cin` and `cout` (for input and output streams, respectively)
- Input file stream usage is like `cin`:

```
int x, y, z;
double a, b, c;

in1 >> x >> y >> z;       // read three integers from the file
in1 >> a >> b >> c;       // read three doubles from the file
```

- Output file stream usage is like `cout`:

```
out1 << "Hello, World\n";  // print "Hello, World" to the file
out1 << "x + y  = " << x + y;      // print a math result to the file
```

Opening a file in 'append mode'

- The default way for opening an output file is to create a brand new file and begin writing from the beginning
    - Note: If another file with the same name already exists, it will be overwritten!
- Existing files can be opened for output, so that the new output is tacked on to the end. This is called *appending*.
- To open a file in *append* mode, we use an extra parameter in the `open()` function:

```
ofstream fout;                      // create file stream
fout.open("file.txt", ios::app); // open file in append mode
```

- There are a number of special constants like this one (`ios::app`). This one will cause a file to be opened for appending .