CSC2430

File I/O part 2

Review: File I/O

- Header file:
  - `#include <fstream>`
- Declaring variables
  - You declare a variable of type `ifstream` for reading or of type `ofstream` for writing
- Associating your file with the variable:
  - You need to either specify the filename in the constructor, or use the `open` method to make the association
- Reading from or writing to file
  - Works nearly the same as console I/O
- Disassociating your file with the variable:
  - If you specified the filename in the constructor let the destructor close it. If you used the `open` method, then call `close` method when you are done with the file

Streams as parameters

- Streams are ALWAYS pass-by-reference (&)
- Example: Function to open files:

```
void openOutputFile (ofstream& fout)
{
    string name;

    cout << "Enter the name of the file (complete path): ";
    getline (cin, name);

    fout.open(name);
    if (fout.fail())
    {
        cout << "Cannot open `" << name << "'\n";
        exit (1);
    }
}
```

Example: writing a line of text

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ofstream fout("greeting.txt");

    if (fout.fail())
    {
        cerr << "Can't open file." << endl;
        return 1;  // ret code 1 indicates error
    }
    fout << "Hello World!" << endl;

    return 0;
}
```

Example: reading a line of text

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
    ifstream fin;
    string line;

    fin.open("greeting.txt");
    if (fin.fail())
    {
        cerr << "Can't open file." << endl;
        return 1; // ret code 1 indicates error
    }
    getline(fin, line);
    cout << line;
    fin.close();
    return 0;
}
```

What can do with your stream?

- For `ofstream`
  - Use <<
  - Use I/O manipulators – don't forget `#include <iomanip>`

- For `ifstream`
  - Use `getline(fin, line)` to read a whole line of text
  - Use >> number to read a number
  - Use >> string to read a sequence of non-whitespace characters
  - Use `fin.get(ch)` to read next character
  - Use `fin.ignore( n, ch )` read up to `n` characters or until it hits `ch`
  - Use `fin.peek()` to return next character without reading it
  - Use `fin.tellg()` to return the current position in the file
  - Use `fin.seekg( pos )` to move to position `pos` in the file

Reading through a file

- Read through the file with getline can be done with simple loop

```
while (getline(fin, line))
    cout << line << endl;    //do something with data
```

- But if you have multiple data items per line or numeric data to read, you will want to use >>
- When using >> to read through a file, you might want to do an initial read before starting loop to "prime" the read

```
fin >> data;
while(!fin.eof())
{
    cout << data << endl;    //do something with data
    fin >> data;
}
```

- This assumes that the last line of file ends with '\n'. What happens if that's not the case?

Your turn…

- Pair up with your neighbor to write this function:
- Write this function called `skipWhite` that reads past any "space" character until the next character to be read is some other character or `EOF`
  - Recall that your parameter MUST be a reference parameter
  - You will want to use `fin.peek();`
  - Use the `isspace(ch)` function in `<cctype>`
  - You can call `fin.ignore()` with no parameters and it will simply read & discard the next character (so long as you are not at EOF)
- 2) Revise the following code to use `skipWhite` function so it works no matter whether or not your file ends with `'\n'`. Expect to do a total rewrite of the logic!

```
fin >> data;
while(!fin.eof())
{
    cout << data << endl;  //do something with data
    fin >> data;
}
```

Solution to exercise

Solution to exercise

```
void skipWhite(ifstream& fin)
{
   int ch;

   while(true)
   {
      ch = fin.peek();
      if ( !isspace(ch))
         break;
      fin.ignore(1);
   }
}
```

```
// Echo the file to console
skipWhite(fin);
while(!fin.eof())
{
    fin >> data;
    cout << data << endl;
    skipWhite(fin);
}
```

Can you mix `getline` and >>?

- What does `getline` do?
  - Read characters into string variable until read a \n (end of line)
  - The \n is discarded (not put into string variable)
- What does >> do when used with a string variable?
  - Reads and discards initial sequence of whitespace characters (blanks, \t tab, \n end of line)
  - Reads sequence non-whitespace characters and put into string variable
  - When it looks ahead and sees a whitespace character, it stops and leaves the whitespace character unread

How could calling skipWhite help?

So, if your file looks like this…

```
Hi
There

```

What happens when each code fragment is run?

```
ifstream fin("yourFile")

getline(fin, line);
fin >> str;
```

```
ifstream fin("yourFile")

fin >> str;
getline(fin, line);
```

```
ifstream fin;
string str, line;
fin.open("afile.txt");
getline(fin, line);
fin >> str;
cout << "line = " << line << endl;
cout << "str  = " << str << endl;
fin.close();

fin.open("afile.txt");
fin >> str;
getline(fin, line);
cout << "line = " << line << endl;
cout << "str  = " << str << endl;
fin.close();
```

```
line = Hi
str  = There
line =
str  = Hi
```

Behind the scenes with `ofstream`

Writing to a file

```
ofstream  fout("myFile");

fout << "Hello World!" << endl;
fout << "We're in CSC 2430";
```

`cout` also buffers output, but it coordinates with `cin` so output gets flushed before cin is read. Why?

Output initially "buffered" in memory

- By default, what you write to an `ofstream` is first saved up in a "buffer" (block of memory).
- Write is delayed until buffer is full, you call "flush", or file is closed.
- Why is this done? Better performance!
- What do you think your file would contain if your program crashes before all the data is flushed to disk?

Eventual "flush" to storage device

Behind the scenes with `ifstream`

Reading from a file

```
ifstream  fin("myFile");
string line;

getline(fin, line);
```

Behind the scenes with `ifstream`

Reading from a file

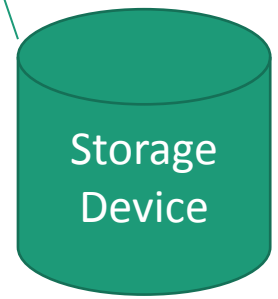
```
ifstream  fin("myFile");
string line;

getline(fin, line);
```

Read here

Buffer containing contents of block read

Block read from storage into buffer



- An `ifstream` object reads a whole block of data from the file into an in memory "buffer"

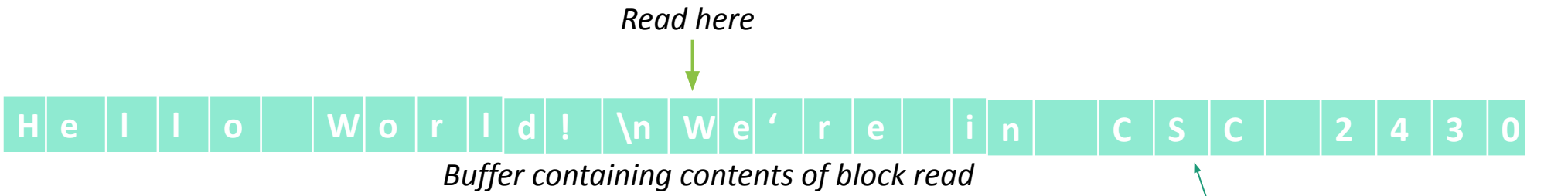
Behind the scenes with `ifstream`

Reading from a file

```
ifstream  fin("myFile");
string line;

getline(fin, line);
```

By default, `cin` reads one line into a buffer. Why doesn't `cin` wait until it gets a full buffer of characters?



- An `ifstream` object reads a whole block of data from the file into an in memory "buffer"
- `getline()` call copies characters from the buffer until sees end of line ('\n') into the "line" variable. Pointer to start of "unread" text is advanced
  - Note: "'\n' is not copied into "line"

What about wide characters?

- Use wifstream and wofstream instead…

- https://github.com/arias-spu/CSC-CPP-Examples