PIC 10A

Lecture 21: File I/O

Reading and Writing to Files

Good news! If you understand cin/cout you are already halfway to understanding reading and writing to files.

cin and cout are examples of stream objects that read and write to the console.

We will define file stream objects for reading and writing from a file.

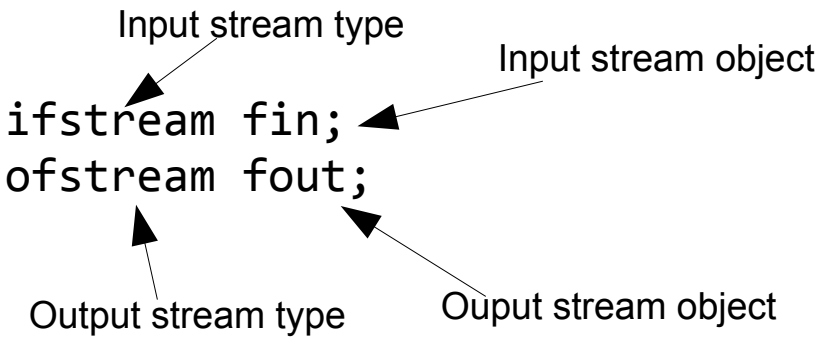
After we define them we use them just like we use cin and cout.

File streams

First thing we need to do is `#include<fstream>`

This is the file stream library that will allow us to create the stream objects that can read and write text files.

To create an file stream objects we write



We can then use this stream to open a file and then read from and write to that file just like cin/cout.

```
int x;
fin >> x; //Read x from the file associated with fin.
fout << x; //Write x to fout's file, could be a different file.
```

Writing to a file

```
#include<fstream> // File stream
using namespace std;

int main()
{

    ofstream fout; //Note how I can pick any name I want for the
                   //object

    fout.open("MyFile.txt");  //Open file named MyFile.txt for
                              //writing.

    fout << "Goodbye cruel world.";

    fout.close(); // Close the file

    return 0;
}
```

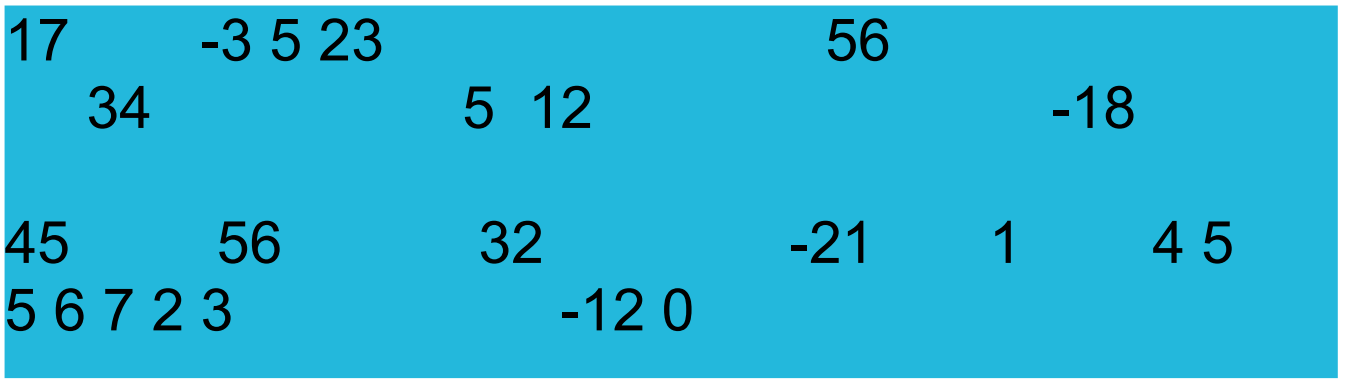
Writing to a file

- If MyFile.txt did not exist, it was created.
- If it did exist, it was overwritten.
- If you want to append your stuff to the end use:

```
fout.open("MyFile.txt", ios::app);
```

Reading from a file

Our goal is to read data from a text file of numbers



numbers.txt

Notice that numbers could be separated by all sorts of white spaces and newlines.

Reading from a file

```
#include<fstream> //Again we have to #include <fstream>
using namespace std;

int main()
{
    ifstream fin;  //This time we declare an ifstream object.
                   //We get to pick the name of the object, but fin
                   //is popular.

    fin.open("MyFile.txt"); //Open file and connect fin object to
                            //this file

    int number;

    fin >> number; //Just like using cin.  number now has the first
                   //integer from the file

    return 0;
}
```

Reading numbers into a vector

We could read numbers into a vector or an array, but if we do not know how many numbers are in the file its probably best to use a vector.

```
int main()
{
    ifstream fin;

    fin.open("MyFile.txt");

    vector<int> MyNumbers;
    int number;

    while(fin >> number)
        MyNumbers.push_back(number);

    fin.close();

    return 0;
}
```

Using a variable file name

Sometimes we want the file name to come from a variable. For example we might ask the user what file they want to open.

```
string file_name;
cin >> file_name;

ifstream fin;
fin.open( filename); // Unfortunately this doesn't work!
```

open( ) is an older function that doesn't recognize strings.

So we have to write

```
fin.open(filename.c_str());
```

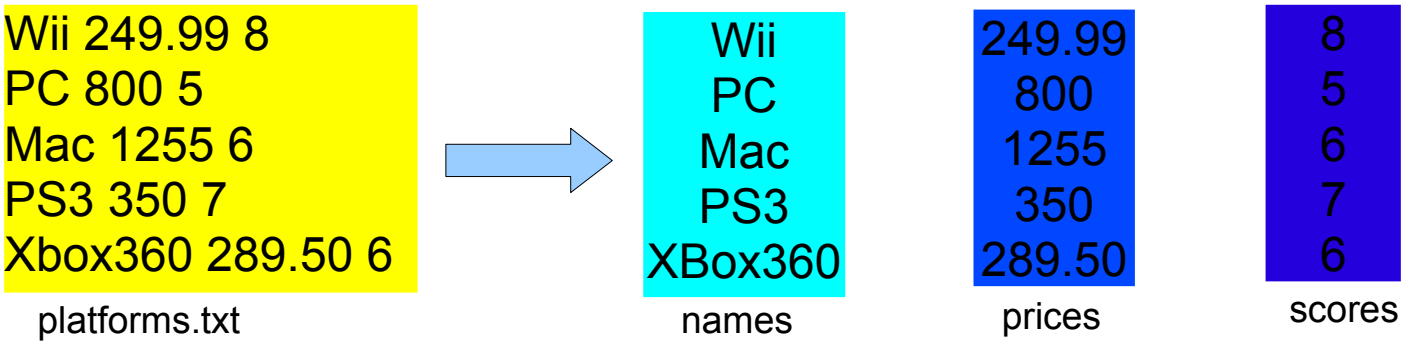
We add c_str( ) to convert the string to a char array.

Example

```
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int main()
{
    string file_name;
    cout << "Enter file name to read: ";
    cin >> file_name;
    ifstream fin;
    fin.open(file_name.c_str( ));
    vector<int> list;
    int x;
    while (fin >> x)
        list.push_back(x);
    fin.close( );
    int ave = average(list);
    cout << "The average is" << ave << "\n";
    return 0;
}
```

Reading file data to parallel vectors

Assume that we have a text file that looks like:



We read each line of platforms.txt into 3 different variables and then push them onto the 3 vectors.

```
string name; double price; int score;

fin >> name >> price >> score;

names.push_back(name);
prices.push_back(price);
scores.push_back(score);
```

Parallel vectors

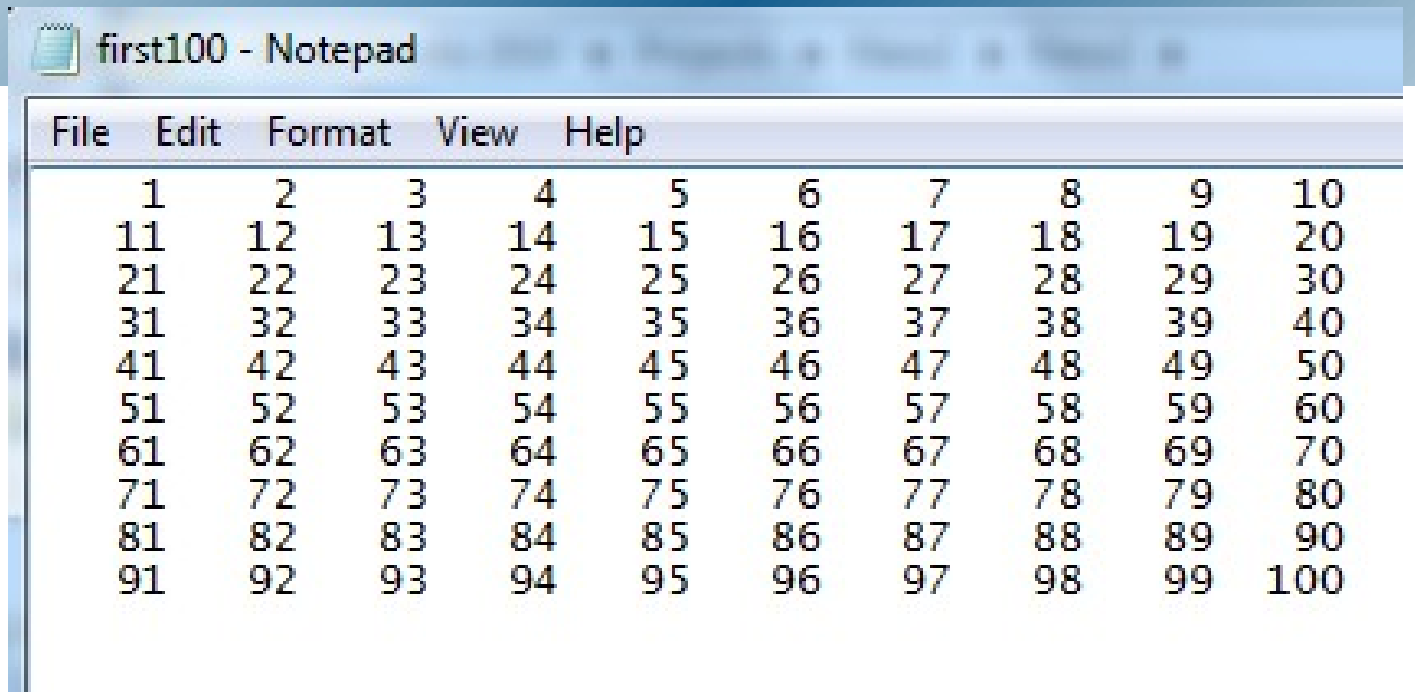
```
ifstream fin;
fin.open("platforms.txt");
vector<string> names;
vector<double> prices;
vector<int> scores;
string score; double price; int score;
while (fin >> name >> price >> score) {
    names.push_back(name);
    prices.push_back(price);
    scores.push_back(score);
}
fin.close( );
for (int j = 0; j < name.size( ); j++)
    cout << name[j] << "\'s ratio is " << price[j] /
            score[j] << "\n";
```

Formatting Output

All the formatting commands like setw and setprecision(n) from <iomanip> carry over to writing to files.

```
#include <fstream>
#include <iomanip>
using namespace std;
int main ( )
{
    ofstream fout;
    fout.open("first100.txt");
    for (int i = 1; i <= 100; i++){
        fout << setw(5) << i;
        if (i%10 == 0)
            fout << "\n";
    }
    fout.close( );
    return 0;
}
```

Formatting output



char data type

A data type we have not discussed in this class is char or character data.

char variable holds a single character.

char is not the same as a string of one character.

string is a complicated data type with member functions.

char is a fundamental data type like ints and doubles.

char actually stores a number corresponding to a character. We never see these numbers because they get always converted back into a letter.

The numbers that the characters correspond to are called ASCII codes. For example A corresponds to 65.

char data type

To declare a character variable we write

```
char my_char;
```

To assign characters to variables we use single quotes.

```
my_char = 'A';
```

We can output the content of the character variable as usual:

```
cout << my_char;
```

Or since we are talking about file I/O today:

```
fout << my_char;
```

Fun with char

```
char letter;

for (int i = 'A'; i <= 'Z'; i++)
{
   letter = (char)i;
   cout << letter << " ";
}
```

The get Function

```
char c;
fin.get(c);
```

This gets a single character from the file and moves the file pointer to the ne
character.

If you have read too far in your file you can move the file pointer back by on
step by:

```
fin.unget( );
```

This is useful for detecting line breaks:

```
if (c=='\n')
```

Encryption example

Our problem is the following: We want to read in a file of text and we want to encrypt it using some encryption algorithm and then we want to output it to a new file.

There are several possible encryption algorithms. In this example we will use a simple encryption method called the Caesar's Cypher

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
N	O	P	Q	R	S	T	U	V	X	Y	Z	W	A	B	C	D	E	F	G	H	I	J	K	L	M

Encryption example

```
ifstream fin;
ofstream fout;

string in_file, out_file;
char letter, encrypted_letter;

cout << "Enter the file to read: ";
cin >> in_file;
cout << "Enter the destination file: ";
cin >> out_file;

fin.open(in_file.c_str( ));
fout.open(out_file.c_str( ));

fin.get(letter);
while (!fin.eof()) {
    encrypted_letter = encrypt(letter);
    fout << encrypted_letter;
    fin.get(letter);
}
fin.close( );
fout.close( );
```

Encryption example

This is the basic encryption function. It has a lot of deficiencies.

It only encrypts properly lower case letters
It does not handle punctuations properly
It does not handle newlines properly

```
char encrypt(char letter)
{
    if (letter ==' ')
        return letter;
    else
    {
        letter -= 'a';
        letter = letter + ('n'-'a');
        letter = letter % 26;
        letter += 'a';
    }
    return letter;
}
```

Detecting line breaks

Suppose our input file has newlines

Jerry: I noticed she's big on the phrase "yada yada."
George: Is "yada yada" bad?
Jerry: No, "yada yada" is good. She's very succinct.

We want our encrypted output to have line breaks at same places.

Wreel: V abgvprq fur'f ovt ba gur cuenfr "lnqn lnqn."
Trbetr: Vf "lnqn lnqn" onq?
Wreel: Ab, "lnqn lnqn" vf tbbq. Fur'f irel fhppvapg.

Since we already are reading character by character we can detect line breaks easily. We just test if our character is a newline. If it is we just return it as it is.

```
if (letter=='\n')
    return letter;
```

Detecting punctuation

What if we want to do the harder case with punctuation?

Lets write a helper function called is_punctuation that returns true if the character is a punctuation character and false if it is not.

```
bool checkPunctuation (char c)
{
   if (c=='.' || c==',' || c=='?' || c==':' || c=='!' || c=='?'
         || c==39 || c=='"')
      return true;
   else
      return false;
}
```

Detecting capitals

Our strategy for uppercase letter is to first detect if we have an upper case letter.

Then we will transform it to a lowercase letter.

Then we will encrypt the letter as normal.

Then we will transform it back into an uppercase letter.

We will use the function is_upper(c) that is already defined for us. We just need to #include< cctype>.

We can also use a predefined function tolower(c) that returns a lowercase version of c.

We will also use a function toupper(c) which returns uppercase version of c.

Final encryption function

```
char encrypt(char letter)
{
    if (checkPunctuation(letter))
        return letter;

    if (letter ==' ')
        return letter;

    else if (letter=='\n')
        return letter;
    else
    {
        letter -= 'a';
        letter = letter + ('n'-'a');
        letter = letter % 26;
        letter += 'a';
    }
    return letter;
}
```

Final main

```
int main()
{
	ifstream fin;
	ofstream fout;

	string in_file, out_file;
	char letter, encrypted_letter;

	bool upper=false;

	cout << "Enter the file to read: ";
	cin >> in_file;
	cout << "Enter the destination file: ";
	cin >> out_file;

	fin.open(in_file.c_str( ));
	fout.open(out_file.c_str( ));

	fin.get(letter);
	while (!fin.eof()) {
		if (isupper(letter))
		{
			upper=true;
			letter=tolower(letter);
		}
		encrypted_letter = encrypt(letter);

		if (upper)
			encrypted_letter = toupper(encrypted_letter);
		fout << encrypted_letter;
		fin.get(letter);
		upper=false;
	}

	fin.close( );
	fout.close( );

	return 0;
}
```