CS 171 Computer Programming 1

Winter 2016

Lab 9 – File I/O

Total 48pts

NAME: Joseph Mulray LAB: 9

Question 1: (2pts)

In this lab, you will be writing a simple text formatting program. As usual, we will be constructing it in stages. Before writing any code, your first step is to download a test file. You can find "the file in the header of this week's lab. It is called `testtext.fmt`. You should save a copy of this file on your computer. If you are using an IDE, you will need to find its working directory and copy your test file there.

What is the first line of the test file?

Answer:

.ce

Question 2: (5pts)

Your first coding step is to write a main program that does the following:

- Ask the user for an input file name
- Open the input file
- For each line in the input file
    - Read the line from the file
    - Call a line processing function (see what this is below)

For this first stage, your line processing function should simply print the lines out to the screen.

Hint/Reminder: Remember there's different ways to read in a single "thing" from a stream vs. reading in an entire line (and there's some quirks).

When you get that working, put your main function in the answer box.

Answer:

```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int main()
{
    string filename, data;

    cout<<"Enter a file name: ";
    cin>>filename;

    ifstream file;
    file.open(filename);

    while(getline(file,data))
    {
    cout<<data<<endl;
    }
```

```
    file.close();

    return 0;

}
```

Question 3 (10pts)

The first formatting step is to put in proper page breaks. Depending on the typeface used, a typical 8.5x11 sheet of paper can hold 66 lines of text. You should format each page with:

- 5 blank lines at the top,
- 56 lines of text,
- two more blank lines,
- a line with a page number,
- and two last blank lines to get to the top of the next page.

The last page will take a little more thought because you will probably have to add additional blank lines to make up the difference between 56 and the number of lines actually printed on the page.

Change your output (which right now is still to the command line) so that above formatting is applied.

Once you have it working properly copy your code into the answer area.

Answer:

```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int main()
{
    string filename, data;
    int count=0;

    cout<<"Enter a file name: ";
    cin>>filename;

    ifstream file;
    file.open(filename);


    int pagenumber=1;


    while(!file.eof())
    {
    if(count<=5)
    {
    cout<<endl;
    }
    if(count>5 && count<=61)
    {
    getline(file,data);
    cout<<data<<endl;
    }
    if(count>61 && count<=63)
    {
    cout<<endl;
    cout<<endl;
    }
    if(count>63 && count<=64)
    {
        cout<<"Page: " <<pagenumber;
```

```
        pagenumber++;
    }
    if(count>64 && count<=66)
    {
        cout<<endl;
    }
    if(count==66)
    {
    count=0;
    }

    count++;

    }
    cout<<"Page: " <<pagenumber;


    cout<<endl;
    cout<<endl;



    file.close();

    return 0;

}
```

Question 4 (8pts)

Change your processing function to look for certain special lines. If you find one, don't print it. Instead, you should call a function specific to that special operation. The first of these special lines you should support is a line consisting of the string `".ce"` which stands for *center*.

When you get that line, your handling function should read the next line of input and print it out centered on an 80 column line. So if you encounter the lines:

```
.ce
Title of This Report
```

the line you print should have 30 spaces before the string "Title of This Report" (so there's 30 spaces on each side of the text plus the 20 characters of the text itself).

This will actually reduce the number of lines of output you generate, because you aren't printing the ".ce" lines.

So for this part of the lab write a function that takes in a string and outputs it centered on the page. And in your main add the logic to call this function when appropriate.

Although your `main()` function has likely changed to detect this case, just copy your new function into the answer area below. We'll ask for your entire program at the end.

Answer:

```
void processor(string filename)
{
    string center=".ce";
    string data;
    int count=0;

    ifstream file;
    file.open(filename);

    int pagenumber=1;


    while(!file.eof())
    {
        if(count<=5)
        {
            cout<<endl;
        }
        if(count>5 && count<=61)
```

```
        {
            getline(file,data);
            if(center.compare(data)==0)
           {
                getline(file,data);
                cout<<"                                    "<<data<<"
"<<endl;
            }
            else
                cout<<data<<endl;
        }
        if(count>61 && count<=63)
        {
            cout<<endl;
            cout<<endl;
        }
        if(count>63 && count<=64)
        {
            cout<<"Page: " <<pagenumber;
            pagenumber++;
        }
        if(count>64 && count<=66)
        {
            cout<<endl;
        }
        if(count==66)
        {
            count=0;
        }

        count++;

    }
    cout<<"Page: " <<pagenumber;


    cout<<endl;
    cout<<endl;


    file.close();


}
```

Question 5 (8pts)

The next formatting command you should support is one for creating new sections in the document. The command is indicated by the special line "`.se`" which uses the next line as the section title. When processing a "`.se`" directive, you should:

- print two blank lines,
- print a line with the section title preceded by a section number,
- and one more blank line before resuming other text output.

You need to generate the section numbers in your code. If the first instance of the section directive looks like:

```
.se
Introduction
```

then the output line should look like (note the two blank lines before the section number/name and the one after it):

```
1. Introduction
```

As a result of the extra blank lines around the section titles, your output will be longer than it was in previous stages.

Like in the previous question, for this part of the lab write a function that takes in a string and an integer and prints out two blanks lines, followed by the integer and a period and space, then the string. And again of course in your main() you'll have to add the logic to call this function when appropriate as well as keep track of what the current section number should be.

Just copy into your answer area the function you created to print a section title (again we'll see the logic to call it when you copy your entire program at the end).

Answer:

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


void processor(string filename)
{
   string section=".se";
   string center=".ce";
   string data;
   int count=0;
   int sectionNum=1;

   ifstream file;
   file.open(filename);

   int pagenumber=1;


   while(!file.eof())
   {
```

```
if(count<=5)
{
   cout<<endl;
}
if(count>5 && count<=61)
{
   getline(file,data);
   if(center.compare(data)==0)
  {
      getline(file,data);
      cout<<"                    "<<data<<"                    "<<endl;
   }
   else if(section.compare(data)==0)
   {
      cout<<endl;
      cout<<endl;
      getline(file,data);
      cout<<sectionNum<<". "<<data<<endl;
      cout<<endl;
      sectionNum++;
   }
   else
      cout<<data<<endl;
}
if(count>61 && count<=63)
{
   cout<<endl;
```

```
        cout<<endl;
    }
    if(count>63 && count<=64)
    {
        cout<<"Page: " <<pagenumber;
        pagenumber++;
    }
    if(count>64 && count<=66)
    {
        cout<<endl;
    }
    if(count==66)
    {
        count=0;
    }

    count++;

}
cout<<"Page: " <<pagenumber;



cout<<endl;
cout<<endl;



file.close();
```

```
}



int main()
{
   string filename;

   cout<<"Enter a file name: ";
   cin>>filename;



   processor(filename);

   return 0;

}
```

Question 6 (5pts)

Now modify your program so that it asks the user for an output file name and prints the output to that file instead of to the screen.

Hint: This most likely will involve changing your functions to accept an output stream as a parameter.

To verify that you got this working, run your program to generate an output file. We created another upload problem in the lab for you to upload the file you generate.

```
void processor(string filename, string outputname)
{
   string section=".se";
   string center=".ce";
   string data;
   int count=0;
   int sectionNum=1;

   ifstream file;
   file.open(filename);

   ofstream outputFile;
   outputFile.open(outputname);

   int pagenumber=1;



   while(!file.eof())
   {
      if(count<=5)
```

```
{
   outputFile<<endl;
}
if(count>5 && count<=61)
{
   getline(file,data);
   if(center.compare(data)==0)
  {
      getline(file,data);
      outputFile<<"                    "<<data<<"                    "<<endl;
   }
   else if(section.compare(data)==0)
   {
      outputFile<<endl;
      outputFile<<endl;
      getline(file,data);
      outputFile<<sectionNum<<". "<<data<<endl;
      outputFile<<endl;
      sectionNum++;
   }
   else
      outputFile<<data<<endl;
}
if(count>61 && count<=63)
{
   outputFile<<endl;
   outputFile<<endl;
```

```
    }
    if(count>63 && count<=64)
    {
        outputFile<<"Page: " <<pagenumber;
        pagenumber++;
    }
    if(count>64 && count<=66)
    {
        outputFile<<endl;
    }
    if(count==66)
    {
        count=0;
    }

    count++;

}
outputFile<<endl;
outputFile<<endl;
outputFile<<"Page: " <<pagenumber;



outputFile<<endl;
outputFile<<endl;
```

```
    file.close();



}
```

Question 7 (10pts)

Copy your entire source code to the answer area below. Here we can verify your logic in the `main()` etc..

Answer:

**This first section header should be numbered 1. It should
paragraph.
And this is a second paragraph in the introduction.
Commands
lines before the section header and one after. It shold**

**include:
– .ce which centers the next line
– .se which makes the next line a section header
Centering
the length of the line that you are centering. Then you
and the length of the line to center. Divide that by 2
before the line that you are centering.
If you do the extra credit part where you handle line
to terminate the current paragraph. However, unlike
doesn't automatically print a blank line after the
directly between two paragraphs with no spacing like
.ce
Or we can print a centered line with a blank line before
The result looks like this:
.ce**

**line before or after and not in the other position.
Sections
This one is Section 4.
Each section header is preceeded by two blank lines and
of the section. As with the centering directive, .se
extra credit part. Unlike the centering case, section
regardless of whether the input file includes blank
.se
Of course, the set of capabilities that are described here
real one would certainly have additional features, such
etc. There would also have to be a system for supporting
that might need to move from one place to another**

**formatter. Typically this will result in the chapter beginning
formatting.
Another important function that we would like to have
itemized and enumerated lists. In effect, we want a
outlines where the formatting software does the
familiar from the <ol> and <ul> tags in HTML.
Filler
on, the remainder of this test file is just junk
couple of page breaks.
asfasdfasdf
asdfasdfas
asdfasdfasdf**

**asdfasdfa
asdfadsfadf**

asdfasdfadsf
asdfadadf

asdfadfadf
asdfasfaf
.ce

Page: 1

asdfasdfasdf
asdfasdfasdf

Page: 2

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


void processor(string filename, string outputname)
{
   string section=".se";
   string center=".ce";
   string data;
   int count=0;
   int sectionNum=1;
```

```
ifstream file;
file.open(filename);

ofstream outputFile;
outputFile.open(outputname);

int pagenumber=1;



while(!file.eof())
{
   if(count<=5)
   {
      outputFile<<endl;
   }
   if(count>5 && count<=61)
   {
      getline(file,data);
      if(center.compare(data)==0)
     {
         getline(file,data);
         outputFile<<"                    "<<data<<"                    "<<endl;
      }
      else if(section.compare(data)==0)
      {
         outputFile<<endl;
```

```
        outputFile<<endl;
        getline(file,data);
        outputFile<<sectionNum<<". "<<data<<endl;
        outputFile<<endl;
        sectionNum++;
    }
    else
        outputFile<<data<<endl;
}
if(count>61 && count<=63)
{
    outputFile<<endl;
    outputFile<<endl;
}
if(count>63 && count<=64)
{
    outputFile<<"Page: " <<pagenumber;
    pagenumber++;
}
if(count>64 && count<=66)
{
    outputFile<<endl;
}
if(count==66)
{
    count=0;
}
```

```
        count++;

    }
    outputFile<<endl;
    outputFile<<endl;
    outputFile<<"Page: " <<pagenumber;



    outputFile<<endl;
    outputFile<<endl;



    file.close();



}



int main()
{
    string filename,outputname;

    cout<<"Enter a file name: ";
    cin>>filename;
```

```
    cout<<"Enter a file output name:";
    cin>>outputname;


    processor(filename,outputname);

    return 0;

}
```