```
/*
 PROGRAM NAME: p2.h

 PROGRAMMER:   James Francis

 CLASS:        CSC 331.001, Fall 2014

 INSTRUCTOR:   Dr. Robert Strader

 DATE STARTED: September 19, 2014

 DUE DATE:     September 23, 2014

 PROGRAM PURPOSE:
 header file used to define the p2 class and include all the libraries to be used in the class.

 VARIABLE DICTIONARY:
 int start1: value representing the first line of the first input file to be appended to the output file
 int end1: value representing the last line of the first input file to be appended to the output file
 int start2: value representing the first line of the second input file to be appended to the output file
 int end2: value representing the first line of the second input file to be appended to the output file

 ADTs: None

 FILES USED: None

 SAMPLE INPUTS: None

 SAMPLE OUTPUTS: None
 ---------------------------------------------------------------------*/


#ifndef p2_p2_h
#define p2_p2_h
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <sstream>

using namespace std;
class p2{

public:

p2(string , string);
void parseInput1(string);
```

```
    44	void parseInput2(string);
    45	void mergeFile(ifstream& ,ofstream& ,int ,int );
    46	const int getStart1();
    47	const int getEnd1();
    48	const int getStart2();
    49	const int getEnd2();
    50	int areLinesValid(int, int);
    51	int isRangeValid();


    52	private:
    53	
    54	int start1;
    55	int end1;
    56	int start2;
    57	int end2;
    58	void setStart1(int);
    59	void setEnd1(int);
    60	void setStart2(int);
    61	void setEnd2(int);
    62	};
    63	#endif

:
printf \\n\\n


cat -b p2.cpp
     1	/*
     2	 PROGRAM NAME: Program 2: File Merge
     3	
     4	 PROGRAMMER:   James Francis
     5	
     6	 CLASS:        CSC 331.001, Fall 2014
     7	
     8	 INSTRUCTOR:   Dr. Robert Strader
     9	
    10	 DATE STARTED: September 19, 2014
    11	
    12	 DUE DATE:     September 23, 2014
    13	
    14	 PROGRAM PURPOSE:
    15	
    16	 This class will accept two strings that contain line ranges and use the parseInput1() &
    17	 parseInput2() functions to set the start and end lines for the merge function to use.
    18	
```

```
VARIABLE DICTIONARY:

int dump: used by the mergeFile() function to iterate through lines not wanted in the output file
int startMerge: value of first line to be written to the output file
int endMerge: value of the last line to be written to the output file
string line: used to hold a line from an input file
ifstream& infile: reference to an input filestream from the calling code
ofstream& outfile: reference to an output filestream from the calling code
string argv: a set of characters that represent the start and end values
             that will be appended from the input file.
string delimeter: a dash, that is used to represent lines between the start
                  and end values.
strong token: used to store a substring of the input
size_t pos: starting position for the substring
string startEnd: used to hold the value of argv

ADTs: none

FILES USED: none


SAMPLE INPUTS:



SAMPLE OUTPUTS:


----------------------------------------------------------------------*/
```

```
#include "p2.h"

p2::p2(string startEnd1, string startEnd2 ){
    //------------------------------------------------------------------------------
    // This is the only constructor for the p2 class. This constructor accepts two
    // strings, that contain the start and end values for the 2 files to be merged.
    //------------------------------------------------------------------------------
    parseInput1(startEnd1);
    parseInput2(startEnd2);
}

void p2::mergeFile(ifstream& infile,ofstream& outfile,int startMerge,int endMerge){
    //------------------------------------------------------------------------------
    // This method will accept an input filestream, output filestream, beginning
    // index, the line of the input file to start the merge, and the line of the
    // input file to end the merge. It will then append the lines of the input file
    // to the output file. This method is used for both input files passed through
    // at command line.
    //------------------------------------------------------------------------------

    string line;
    int dump = 1;
    while(dump<startMerge){
        getline(infile,line);
        dump++;
    }

    while(startMerge<=endMerge){
        getline(infile,line);
        outfile<<line+"\n";
        startMerge++;
    }


}

void p2::parseInput1(string argv){
    //------------------------------------------------------------------------------
    // This function accepts a string, it is assimed the string contains 2 integers
    // delimited by a '-'. This function then tries to set the separated values
    // as the beginning and ending lines for the first file.
    //------------------------------------------------------------------------------
    string startEnd = argv;
    string delimiter = "-";

    size_t pos = 0;
    string token;
    while ((pos = startEnd.find(delimiter)) != string::npos) {
        token = startEnd.substr(0, pos);
        try {
            stringstream ss(token);
            int value;
            ss>>value;
            setStart1(value);
        } catch (int e) {
            cout << "Invalid start value for File1, please try again.\n";
            exit(-7);
        }
        startEnd.erase(0, pos + delimiter.length());
    }
    try {
        stringstream ss(startEnd);
        int value;
        ss>>value;
        setEnd1(value);

    } catch (int e) {
        cout << "Invalid end value for File1, please try again.\n";
        exit(-6);

    }

}
void p2::parseInput2(string argv){
    //------------------------------------------------------------------------------------
    // This function accepts a string, it is assimed the string contains 2 integers
    // delimited by a '-'. This function then tries to set the separated values
    // as the beginning and ending lines for the second file.
    //------------------------------------------------------------------------------------
    string startEnd = argv;
    string delimiter = "-";

    size_t pos = 0;
    string token;
    while ((pos = startEnd.find(delimiter)) != string::npos) {
        token = startEnd.substr(0, pos);
        try {
            stringstream ss(token);
            int value;
            ss>>value;
            setStart2(value);
        } catch (int e) {
            cout << "Invalid start value for File2, please try again.";
            exit(-5);
        }
        startEnd.erase(0, pos + delimiter.length());
    }
    try {
        stringstream ss(startEnd);
        int value;
        ss>>value;
        setEnd2(value);

    } catch (int e) {
        cout << "Invalid end value for File2, please try again.";
        exit(-4);
    }


}
int p2::areLinesValid(int n, int m){
    //-----------------------------------------------------------------------------
    // This method accepts integer values from the calling code and compares those
    // with the end values for each of the input files' entered end lines.
    // If either end value entered at command line is greater than the passed
    // integers then the line count for one of the files is exceeded and a 0 is
    // returned to the calling code.
    //-----------------------------------------------------------------------------

    if (end1 > n  || end2 > m) {
        return 0;
    }
    else return 1;
}
int p2::isRangeValid(){
    //-----------------------------------------------------------------------------
    // This method compares the proposed start and end lines for both files.
    // if either of the start values exceed their respective end values then a 0 is
    // returned to the calling code.
    //-----------------------------------------------------------------------------

    if (start1 > end1  || start2 > end2) {
        return 0;
    }
    else return 1;
}
//-----------------------------------------------------------------------------
// Setters and Getters follow
//-----------------------------------------------------------------------------
void p2::setStart1(int n){
    if (n>0) {
        start1 = n;
    }
}
```

```
   185 void p2::setEnd1(int n){
   186     if (n>0) {
   187         end1 = n;
   188     }
   189 }

   190 void p2::setStart2(int n){
   191     if (n>0) {
   192         start2 = n;
   193     }
   194 }

   195 void p2::setEnd2(int n){
   196     if (n>0) {
   197         end2 = n;
   198     }
   199 }


   200 const int p2::getStart1(){
   201     return p2::start1;
   202 }
   203 const int p2::getStart2(){
   204     return p2::start2;
   205 }
   206 const int p2::getEnd1(){
   207     return p2::end1;
   208 }
   209 const int p2::getEnd2(){
   210     return p2::end2;
   211 }
:
printf \\n\\n


cat -b p2Test.cpp
     1 /*
     2  PROGRAM NAME: Program 2: File Merge
     3
     4  PROGRAMMER:   James Francis
     5
     6  CLASS:        CSC 331.001, Fall 2014
     7
     8  INSTRUCTOR:   Dr. Robert Strader
     9
    10  DATE STARTED: September 19, 2014
```

```

DUE DATE:     September 23, 2014

PROGRAM PURPOSE:
This text class is used to test p2. This class accepts command line arguments that are used to
instantiate the p2 class. This class accepts 5 arguments in the form of:

p2 infile.dat startLine-endLine infile2.dat startLine-endLine outFile.dat

where infile and infile2 are the source files, startLine is the first line from the proceeding infile
to be appended to the outFile, and endLine is the last line from the proceeding infile to be appended
to the outfile.

The file outputs outfile.

VARIABLE DICTIONARY:


ADTs: none

FILES USED: prog2a.dat prog2b.dat outputfile.dat


SAMPLE INPUTS:

(command line)
p2 prog2a.dat 1-5 prog2b.dat 3-7 outfile.dat

(prog2a.dat)
This is the first line
and this is the second line
and this is the third line
and the fourth
and the fifth
and here is the sixth and

(prog2b.dat)
This is the 1st line of many.
Here is the 2nd of several,
and the 3rd followed by
the 4th as well as the
5th which is the last in this paragraph
A new paragraph has the 6th, followed
by the 7th and
the 8th
and 9th closing with
```

```
 
 SAMPLE OUTPUTS:(to outfile.dat)
 
 This is the first line
 and this is the second line
 and this is the third line
 and the fourth
 and the fifth
 and the 3rd followed by
 the 4th as well as the
 5th which is the last in this paragraph
 A new paragraph has the 6th, followed
 by the 7th and
 
 --------------------------------------------------------------------*/

#include "p2.h"

int checkEndValue(string);

int main(int argc, const char * argv[]){

    if (argc < 5) {

        // Informs the user how to run the program if they enter fewer than 5 arguments

        cerr << "Invalid number of arguments"<< endl;
        cerr << "Please ensure you have entered a command line argument in the following format:" << endl;
        cerr << endl;
        cerr << "fileMerge inputFile1 lineStart-lineEnd inputFile2 lineStart-lineEnd outputFile"<< endl;
        cerr << endl;
        cerr << "Example: fileMerge infil.dat 100-200 infil2.dat 150-300 outfile.dat\n"<< endl;
        return -3;
    }

    string file1 = argv[1];
    ifstream infile1;
    infile1.open(file1.c_str());
        if (!infile1.is_open()) {
           cerr << "Error opening file: "<<argv[1]<< "\nplease be sure the paths to the source files are correct and try again.\n";
            return -2;
        }

    string file2 = argv[3];
    ifstream infile2;
```

```cpp
    infile2.open(file2.c_str());
        if (!infile2.is_open()) {
            cerr << "Error opening file: "<<argv[3]<< "\nplease be sure the paths to the source files are correct and try again.\n";
            return -1;
        }
    string startEnd1 = argv[2];
    string startEnd2 = argv[4];


    p2 merger = p2(startEnd1, startEnd2);

    if (merger.isRangeValid()==0) {
        cerr << "Start line cannot exceed end line. Check your inputs and try again.\n";
        return 0;
    }else if(merger.areLinesValid(checkEndValue(file1), checkEndValue(file2))==0){
        cerr << "Start line must be greater than 0 and the end line cannot exceed the number of lines in a file, check your inputs and try again.\n";
        return 1;
    }
    string file3 = argv[5];
    ofstream myfile3;
    myfile3.open(file3.c_str(), ios::out);
    //Opens an output filestream with an appending flag
    merger.mergeFile(infile1, myfile3, merger.getStart1(), merger.getEnd1());
    //Merges the first input file into the output file

    merger.mergeFile(infile2, myfile3, merger.getStart2(), merger.getEnd2());
    //Appends the second input file into the output file, after the first file

    infile1.close();
    infile2.close();
    myfile3.close();

    return 2;
}
int checkEndValue(string filename){
    //---------------------------------------------------------------------------
    // This method accepts a filename and opens a stream, separate from the stream
    // used in the merger, and counts the lines in the passed filename, and returns
    // that count as an integer to the calling code.
    //-----------------------------------------------------------------------------
    ifstream file;
    file.open(filename.c_str());
    int j = 0;
    string line;
    while(getline(file, line)){
        j++;
    }
    file.close();
    return j;
}
:
g++ p2.cpp p2Test.cpp -o p2
:
p2 prog2a.dat 1-3 prog2b.dat 5-8 outfile.dat
```

```
cat -b prog2a.dat
     1	This is the first line
     2	and this is the second line
     3	and this is the third line
     4	and the fourth
     5	and the fifth
     6	and here is the sixth and
     7	this is the seventh with
     8	the eighth following and the
     9	ninth line here.
    10	The tenth starts a new paragraph
    11	with the eleventh and
    12	the twelfth followed by the
    13	thirteenth line, luck thirteenth
    14	as well as the fourteenth
    15	and the fifteenth and of course
    16	the sixteenth and the
    17	seventeenth.  Are there
    18	any more, yes the eighteenth
    19	and the nineteenth and finally the
    20	twentieth.
```

```
cat -b prog2b.dat
     1	This is the 1st line of many.
     2	Here is the 2nd of several,
     3	and the 3rd followed by
     4	the 4th as well as the
     5	5th which is the last in this paragraph
     6	A new paragraph has the 6th, followed
     7	by the 7th and
     8	the 8th
     9	and 9th closing with
    10	the 10th.
    11	For the next info the 11th will
    12	lead followed by the 12th and
    13	the 13th with the
    14	14th not far behind and ending
    15	with the 15th.
    16	Finally the 16th
    17	17th and
    18	18th with the
    19	19th and the
    20	20th rounding out the document.


cat -b outfile.dat
     1	This is the first line
     2	and this is the second line
     3	and this is the third line
     4	5th which is the last in this paragraph
     5	A new paragraph has the 6th, followed
     6	by the 7th and
     7	the 8th


date
```