Assignment: Program5
Author: Michael Lapan

```
----ALERT.H---

#include<string>
#include "dateTime.h"
#include "fips.h"
using namespace std;

/********************************************//**
* \class alert
*
* alert brief description
*                                        this class manages all of the alert objects
*
*this class populates the list of alert objects,
*sorts them by fips code, outputs them sorted or
*non sorted, and also can find by fips code and
* output all info about that fips code.
*sorting is done by quick sort
***********************************************/
class Alert
{
private:
        string dateTimeString; //!< string to hold unparsed date and time
        fips fipsCode;  //!< object to hold info about fips code
        string code;  //!< holds the weather code
        string pop;  //!< hold the population
        dateTime dateTimeS;  //!< dateTime object for the starting date and time
        dateTime dateTimeE;  //!<  dateTime object for the ending date and time

public:


        /********************************************//**
        *getdateTimeString function deff
        *@return [out] - returns the dateTimeString
        ***********************************************/
        string getdateTimeString();

        /********************************************//**
        *getfips function deff
        *@return [out] - returns the fips code object
        ***********************************************/
        fips getfips();

        /********************************************//**
        *getCode function deff
        *@return [out] - returns the weather code
        ***********************************************/
        string getCode();

        /********************************************//**
        *getDateTimeS function deff
        *@return [out] - returns the starting dateTime object
        ***********************************************/
        dateTime getDateTimeS();

        /********************************************//**
        *getDateTimeE function deff
        *@return [out] - returns the ending dateTime object
```

```
***********************************************/
dateTime getDateTimeE();

/*******************************************//**
*getParsedDateTime function deff
*@return [out] - returns the parsed dateTime object
***********************************************/
dateTime getParsedDateTime();

/*******************************************//**
*getPop function deff
*@return [out] - returns the population
***********************************************/
string getPop();


/*******************************************//**
* function deff
* @param[in] toSet Is the varible
***********************************************/
void setdateTimeString(string toSet);


/*******************************************//**
*setCode function deff
* @param[in] toSet Is the varible code will be set to
***********************************************/
void setCode(string toSet);

/*******************************************//**
*setDateTimeS function deff
* @param[in] toSet Is the varible dateTimeS (s is for start)
*will be set to
***********************************************/
void setDateTimeS(dateTime toSet);

/*******************************************//**
* setDateTimeE function deff
* @param[in] toSet Is the varible dateTimeE (e is for end)
*will be set to
***********************************************/
void setDateTimeE(dateTime toSet);

/*******************************************//**
* setParsedDateTime function deff
* @param[in] toSet Is the object parsedDateTime
*will be set to
***********************************************/
void setParsedDateTime(dateTime toSet);

/*******************************************//**
*setPop function deff
* @param[in] toSet Is the varible population will
*be set to
***********************************************/
void setPop(string toSet);
```

```
        /*******************************************//**
        *writeAlert function deff
        *this quite franksl writes one alert, from
        *all of the objects
        **********************************************/
        void writeAlert();

        /*******************************************//**
        *parseDateTime function deff
        *parses the dateTime string from a number like
        *201502121300 to - 2015, feburary, 12, 1:00pm
        *is all done with math. I divide and truncate
        *toget each number.
        *@param[in] se this number will determin weather
        *this datetime is the starting dateTime or the
        *ending dateTime
        **********************************************/
        void parseDateTime(int se);

        /*******************************************//**
        *parseCode function deff
        *parse code parses a weather code like wws
        *into something readable by humans. EG.
        *wws = Winter Storm Warning
        **********************************************/
        void parseCode();

        /*******************************************//**
        *popFips function deff
        *finds and sets the population of a defined fips
        *code
        *@param[in] toParse - the fips code to lookup
        *and get and set pop of
        **********************************************/
        void popFips(string toParse);

        /*******************************************//**
        *getPopByFips function deff
        *finds and set the population of a fips code
        *from the file.
        **********************************************/
        void getPopByFips();

        /*******************************************//**
        *searchFipsCode function deff
        *searches for a fips code and then displays info
        *about fips code
        *@param[in] toFind - is the fips code to look for
        **********************************************/
        void searchFipsCode(string toFind);
};
```

```
----------------AlertList.h------------

#include<string>
#include "Alert.h"
using namespace std;

/*******************************************//**
* \class AlertList
*
* AlertList brief description
*                                         this class manages all of the alert objects
*
*this class populates the list of alert objects,
*sorts them by fips code, outputs them sorted or
*non sorted, and also can find by fips code and
* output all info about that fips code.
*sorting is done by quick sort
**********************************************/
class AlertList
{
private:
        int listSize; //!< is the size of the list? not neededD?
        Alert list[15]; //!<  array of alert objects
public:

        /*******************************************//**
        *getListSize function deff
        *this function returns the size of the listSize
        *varible
        **********************************************/
        int getListSize();
        // Alert getList();


        /*******************************************//**
        *setListSize function deff
        *@param[in] toSet sets the size of the list manualy
        **********************************************/
        void setListSize(int toSet);


        /*******************************************//**
        *popList function deff
        *this function populates the list of alert objects
        *fills all alert objects with all the info they
        *they need to make their alert
        **********************************************/
        void popList();


        /*******************************************//**
        *getFileSize function deff
        *this function returns the size of the alerts.txt
        *file. witch is how many alert objects their are.
        **********************************************/
        int getFileSize();
        /*********************************************//**
        *writeList function deff
        *writeList writes the unsorted list
        ************************************************/
        void writeList();

        /*********************************************//**
        *writeSortedListByFips function deff
        *writes the sorted list of alert objects.
        *the list is sorted by fips code
        ************************************************/
        void writeSortedListByFips();

        /*********************************************//**
        * Quicksort, taken from the internet, modified by
        *Michael LaPan to sort an array of objects by the
        *fips code object in the array object.
        * @param [in,out] a - The array to be sorted.
        * @param [in] first - The start of the sequence to be sorted.
        * @param [in] last - The end of the sequence to be sorted.
        ************************************************/
        void quickSort(Alert a[], int first, int last);

        /*********************************************//**
        * Find and return the index of pivot element.
        * @param [in, out] a - The array.
        * @param [in] first - The start of the sequence.
        * @param [in] last - The end of the sequence.
        * @return [out]- the pivot element
        ************************************************/
        int pivot(Alert a[], int first, int last);

        /*********************************************//**
        * Swap the parameters.
        * @param [in,out] a - The first parameter.
        * @param [in,out] b - The second parameter.
        ************************************************/
        void swap(Alert& a, Alert& b);

        /*********************************************//**
        *findFips function deff
        *finds the fips code and outputs the county, state
        *fips code, and population for the fipscode.
        * @param [in] FIPS fips code to find
        ************************************************/
        void findFips(string FIPS);
};
```

```
----------------dateTime.h-p-----------------
#include <string>
using namespace std;

/*******************************************//**
* \class dateTime
*
* dateTime brief description
*                               this class hold the date and military time
*
*this class holds the month, day, year, and time
* for a given date. it can also parse the month
* in to its worded format, and change the time
* from military to normal
***********************************************/

class dateTime
{
private:
        int month; //!< holds the months number
        int day; //!< holds the day number
        int year; //!< holds the year
        int hr; //!< holds only the hour of the time
        int min; //!<  holds only the minute of the time

        string monthP; //!< holds the month changed to worded format
        string timeP; //!< holds the time changed from military time

public:

        /*******************************************//**
        * getMonth function deff
        * returns the numberd month
        *@return [out] - returns the numbered month
        ***********************************************/
        int getMonth();

        /*******************************************//**
        * getDay function deff
        * returns the numbered day
        *@return [out] - returns the day
        ***********************************************/
        int getDay();

        /*******************************************//**
        * getYear function deff
        * returns the year
        *@return [out] - returns the year
        ***********************************************/
        int getYear();

        /*******************************************//**
        * getHr function deff
        * returns the hours in military time
        *@return [out] - returns the hr in military time
        ***********************************************/
        int getHr();
```

```
/********************************************//**
* getMin function deff
* returns the mins
*@return [out] - returns the minutes
***********************************************/
int getMin();

/********************************************//**
* getTimeP function deff
* returns the parsed time, changes time from
*military time to normal
*@return [out] - returns the parsed time
***********************************************/
string getTimeP();

/********************************************//**
* getMonthP function deff
*returns the parsed month, changes the month from
* a number to worded format
*@return [out] - returns the parswed month
***********************************************/
string getMonthP();

/********************************************//**
*  setMonth function deff
*sets the month for the object.
* @param[in] toSet Is the varible month will be set to
***********************************************/
void setMonth(int toSet);

/********************************************//**
*  setDay function deff
*sets the day for the object.
* @param[in] toSet Is the varible day will be set to
***********************************************/
void setDay(int toSet);

/********************************************//**
*  SetYear function deff
*sets the year for the object.
* @param[in] toSet Is the varible year will be set to
***********************************************/
void SetYear(int toSet);

/********************************************//**
*  setHr function deff
*sets the hour for the object.
* @param[in] toSet Is the varible hr will be set to
***********************************************/
void setHr(int toSet);

/********************************************//**
*  setMin function deff
*sets the minute for the object.
* @param[in] toSet Is the varible min will be set to
***********************************************/
void setMin(int toSet);
```

```
        /*******************************************//**
        * parseMonth function deff
        *changes the month from a number to words.
        *throws numbered month into switch. then sets
        *the parsed month to monthP
        **********************************************/
        void parseMonth();

        /*******************************************//**
        * parseTime function deff
        *changes the time from military time into normal
        *time. if hours is over 12 it subtracts 12
        *to get time in the afternoon, if the number is
        * 0 after 12 is selected. it sets the hour
        *to noon. this also combines hours and minutes
        *into one string.
        **********************************************/
        void parseTime();

        //dateTime(double m, double d, double y);
};
```

```
---------fips.h-------------
#include <string>
using namespace std;

/*******************************************//**
*\class fips
*
* fips brief description.
*        this class hold info about the fipsCode
*
*this class hold info about the fips code, like
*the fipsCode, the county for that fipsCode,
*and the state the fips code is in.
**********************************************/
class fips
{
private:
        string fipsCode; //!< Holds the Fips code
        string county; //!< holds the county
        string state; //!< holdes the stae its in
public:

        /*******************************************//**
        * getFipsCode function deff
        *return the fips code for the object.
        *@return [out] - returns the fipscode
        **********************************************/
        string getFipsCode();
        /*******************************************//**
        * getCounty function deff
        *return the county for the object.
        *@return [out] - returns the county
        **********************************************/
        string getCounty();
        /*******************************************//**
        *  getState function deff
        *return the state for the object.
        *@return [out] - returns the state
        **********************************************/
        string getState();

        /*******************************************//**
        *  setFipsCode function deff
        *sets the fips code for the object.
        * @param[in] toSet Is the varible fipsCode will be set to
        **********************************************/
        void setFipsCode(string toSet);
        /*******************************************//**
        *   setCounty function deff
        *sets the county for the object.
        * @param[in] toSet Is the varible county will be set to
        **********************************************/
        void setCounty(string toSet);
        /*******************************************//**
        *   setState function deff
        *sets the state for the object.
        * @param[in] toSet Is the varible state will be set to
        **********************************************/
        void setState(string toSet);

        /*********************************************//**
        * fipsToInt function deff
        * changes the fips code from a string to an int
        ************************************************/
        int fipsToInt();
};
```

```
-------------------alert.cpp-------------------
#include<string>
#include <time.h>
#include <iostream>
#include <sstream>
#include <math.h>
#include <cmath>
#include <fstream>
#include "Alert.h"

		/*******************************************//**
		* **NOTE**
		* everything that has been clearly defined
		*through comments in the header will not be
		*redifined here. but some functions will have
		*a little extra if they do something either
		*hard to follow, or just need to clear them
		*up
		**********************************************/
string Alert::getdateTimeString()
{
		return dateTimeString;
}

fips Alert::getfips()
{
		return fipsCode;
}
dateTime Alert::getParsedDateTime()
{
		return dateTimeS;
}
dateTime Alert::getDateTimeS()
{
		return dateTimeS;
}
dateTime Alert::getDateTimeE()
{
		return dateTimeE;
}
string Alert::getCode()
{
		return code;
}
string  Alert::getPop()
{
		return pop;
}

void Alert::setDateTimeS(dateTime toSet)
{
		dateTimeS = toSet;
}
void Alert::setDateTimeE(dateTime toSet)
{
		dateTimeE = toSet;
}
void Alert::setdateTimeString(string toSet)
```

```
{
        dateTimeString = toSet;
}


void Alert::setParsedDateTime(dateTime toSet)
{
        dateTimeS = toSet;
}
void Alert::setCode(string toSet)
{
        code = toSet;
}
void Alert::setPop(string toSet)
{
        pop = toSet;
}
void Alert::writeAlert()
{
        cout << code << " for " << fipsCode.getCounty() << ", " << fipsCode.getState() << endl
                << dateTimeS.getMonthP() << " " << dateTimeS.getDay() << ", " << dateTimeS.getTimeP()
                << " - " << dateTimeE.getMonthP() << " " << dateTimeE.getDay() << ", " << dateTimeE.getTimeP() << endl
                << "Population Impact: " << pop << endl;
}
        /*******************************************//**
        *parse date time parses out a single string
        *that contains the year,month,day,milltime in
        *that order into something readable by humans.
        *how i do it is i take the number, divid it by
        *a decimal. EX(201502121300 turns into 2015.02121300).
        *i then truncate off the decimal. and boom i have
        *the year. next i take that year, times it by a number.
        *(2015.00000000 * 100000000 = 201500000000). i then
        *take that and minus that from the start time. then
        *i just repete the process again and again untill,
        *everything is parsed out.
        ***********************************************/
void Alert::parseDateTime(int se)
{
        stringstream ss;

        double m = 0, d = 0, tm = 0, th = 0, y = 0;
        int temp;
        double dateTimeStart = 0.0;
        ss << dateTimeString;
        ss >> dateTimeStart;

        y = trunc(dateTimeStart * .00000001);

        dateTimeStart -= y * 100000000;
        m = trunc(dateTimeStart * .000001);

        dateTimeStart -= m * 1000000;
        d = trunc(dateTimeStart * .0001);

        dateTimeStart -= d * 10000;
        th = trunc(dateTimeStart * .01);
```

```
		dateTimeStart -= th * 100;
		tm = trunc(dateTimeStart);
		if (se == 0)
		{
			dateTimeS.setDay(d);
			dateTimeS.setMonth(m);
			dateTimeS.SetYear(y);
			dateTimeS.setHr(th);
			dateTimeS.setMin(tm);
			dateTimeS.parseMonth();
			dateTimeS.parseTime();
		}
		else if (1)
		{
			dateTimeE.setDay(d);
			dateTimeE.setMonth(m);
			dateTimeE.SetYear(y);
			dateTimeE.setHr(th);
			dateTimeE.setMin(tm);
			dateTimeE.parseMonth();
			dateTimeE.parseTime();
		}

		ss.clear();
}
		/*******************************************//**
		*parses WWS or color code in to the alert
		*i loop through the file either looking for my color
		*or looking for the code (with out the_).
		*once found excract data after the code, and
		*make my alert sentance from it.
		***********************************************/
void Alert::parseCode()
{
		int  i = 0, q = 0;
		ifstream inFile("warningList.txt");
		string inputLine;

		if (inFile.fail())	// Test for file existence
		{
			cout << "Problem opening file";
			system("pause");
			exit(-1);
		}

		//priming read
		string nonColor, isColor, temp, WAY;
		//if not a color parse the warning prfix
		if (code != "RED" && code != "ORANGE"
			&& code != "YELLOW" && code != "BLUE"
			&& code != "GREEN")
		{
			temp = code[0];
			code.erase(0, 1);
			if (temp == "W")
			{
				WAY = "Warning";
			}
```

```
				else if (temp == "A")
				{
					WAY = "Watch";
				}
				else if (temp == "Y")
				{
					WAY = "Advisory";
				}
			}

			while (!inFile.eof())
			{
				getline(inFile, inputLine); //continuation read
				stringstream iss(inputLine);

				while (getline(iss, inputLine, '_'))
				{
					//substr is used to skip either the _ or the color and
					//get the data behind it.
					 string::size_type pos = inputLine.find(code);
					if (pos !=  string::npos)
					{
						if (code == "RED" || code == "ORANGE"
							|| code == "YELLOW" || code == "BLUE"
							|| code == "GREEN")
						{
							isColor = inputLine.substr(pos + 8);
						}
						else
						{
							nonColor = inputLine.substr(pos + 3);
						}
					}
				}
				i++;
			}
			inFile.close();
			if (isColor.empty())
			{
				if (WAY.empty())
				{
					code = nonColor;
				}
				else
				{
					code = (nonColor + " " + WAY);
				}
			}
			else
			{
				code = isColor;
			}
		}
			/*******************************************//**
			*populates the fips code object looking for input
			*as criteria
			**********************************************/
```

```
void Alert::popFips(string toParse)
{
	ifstream inFile("fipsCounty.txt");
	string inputLine;

	if (inFile.fail())	// Test for file existence
	{
		cout << "Problem opening file";
		system("pause");
		exit(-1);
	}

	//priming read
	string countyF, stateF, fipsCodeF;

	while (!inFile.eof())
	{
		getline(inFile, inputLine); //continuation read
		stringstream iss(inputLine);
		if (inputLine.find(toParse) != string::npos)
		{
			int i = 0;
			inputLine = inputLine.erase(0, 6);
			while (getline(iss, inputLine, ','))
			{
				if (i == 0)
				{
					countyF = inputLine.erase(0, 6);
				}
				else if (i == 1)
				{
					stateF = inputLine;
				}

				i++;
			}
		}
	}
	fipsCode.setCounty(countyF);
	fipsCode.setFipsCode(toParse);
	fipsCode.setState(stateF);
	inFile.close();
	getPopByFips();
}
	/*******************************************//**
	*gets the population by useing the fips code
	***********************************************/
void Alert::getPopByFips()
{
	ifstream inFile("popcounty.txt");
	string inputLine;

	if (inFile.fail())	// Test for file existence
	{
		cout << "Problem opening file";
		system("pause");
		exit(-1);
	}
```

```
        while (!inFile.eof())
        {
                getline(inFile, inputLine); //read
                stringstream iss(inputLine);
                //loop through file remove ',' and find pop linked with fips code
                if (inputLine.find(fipsCode.getFipsCode()) != string::npos)
                {
                        int i = 0;
                        while (getline(iss, inputLine, ','))
                        {
                                if (i == 1)
                                {
                                        pop = inputLine;
                                }
                                i++;
                        }
                }
        }
        inFile.close();
}
        /********************************************//**
        *searches for a fips code specified by the user
        *loops through file removeing ',' and looking for
        *correct fips code
        ***********************************************/
void Alert::searchFipsCode(string toFind)
{
        int   q = 0;
        ifstream inFile("fipsCounty.txt");
        string inputLine;

        if (inFile.fail())        // Test for file existence
        {
                cout << "Problem opening file";
                system("pause");
                exit(-1);
        }

        //priming read
        string countyF, stateF;
        bool found = false;

        while (!inFile.eof())
        {
                getline(inFile, inputLine); //continuation read
                stringstream iss(inputLine);
                if (inputLine.find(toFind) != string::npos)
                {
                        found = true;
                        int i = 0;
                        inputLine = inputLine.erase(0, 6);
                        while (getline(iss, inputLine, ','))
                        {
                                if (i == 0)
                                {
                                        countyF = inputLine.erase(0, 6);
                                }
                                else if (i == 1)
                                {
                                        stateF = inputLine;
                                }

                                i++;
                        }
                }
        }
        if (found)
        {
                fipsCode.setCounty(countyF);
                fipsCode.setFipsCode(toFind);
                fipsCode.setState(stateF);
                getPopByFips();
        }
        inFile.close();
}
```

```
--------------alertList.cpp-----------------

#include<string>
#include <iostream>
#include <stdlib.h>
#include <fstream>
#include <sstream>
#include "AlertList.h"
using namespace std;



	/*********************************************//**
	* **NOTE**
	* everything that has been clearly defined
	*through comments in the header will not be
	*redifined here. but some functions will have
	*a little extra if they do something either
	*hard to follow, or just need to clear them
	*up
	************************************************/


	/*********************************************//**
	*getFileSize loops throught the file,
	*counts and counts the lines in the file
	*@return [out] - returns the number of lines
	************************************************/

int AlertList::getFileSize()
{
	int i = 0;
	ifstream inFile("alerts.txt");
	string inputLine;

	if (inFile.fail())	// Test for file existence
	{
		cout << "Problem opening file";
		system("pause");
		exit(-1);
	}

	//priming read
	getline(inFile, inputLine);
	while (!inFile.eof())
	{
		getline(inFile, inputLine); //continuation read
		i++;
	}
	return i;
}

	/*********************************************//**
	*populates the list of Alert objects
	*from the alert file.
	************************************************/
```

```
void AlertList::popList()
{
	int size = getFileSize(), i = 0, q = 0;
	ifstream inFile("alerts.txt");
	string inputLine;

	if (inFile.fail())	// Test for file existence
	{
		cout << "Problem opening file";
		system("pause");
		exit(-1);
	}
	//priming read

	while (!inFile.eof())
	{
		getline(inFile, inputLine); //continuation read

		stringstream iss(inputLine);
		//loop through looking for the comma and splitting the data
		while (getline(iss, inputLine, ','))
		{
			if (q == 0)
			{
				list[i].popFips(inputLine);
			}
			else if (q == 1)
			{
				list[i].setdateTimeString(inputLine);
				list[i].parseDateTime(0);
			}
			else if (q == 2)
			{
				list[i].setdateTimeString(inputLine);
				list[i].parseDateTime(1);
			}
			else if (q == 3)
			{
				list[i].setCode(inputLine);
				list[i].parseCode();
			}
			q++;
		}
		q = 0;
		i++;
	}
}

void AlertList::writeList()
{
	int size = getFileSize();
	for (int i = 0; i < size; i++)
	{
		list[i].writeAlert();
		system("pause");
	}
}
```

```
void AlertList::writeSortedListByFips()
{
	quickSort(list, 0, 6);
	int size = getFileSize();
	for (int i = 0; i < size + 1; i++)
	{
		list[i].writeAlert();
		system("pause");
	}
}

/**
* Quicksort, taken from the internet, modified by
*Michael LaPan to sort an array of objects by the
*fips code object in the array object.
* @param a - The array to be sorted.
* @param first - The start of the sequence to be sorted.
* @param last - The end of the sequence to be sorted.
*/
void AlertList::quickSort(Alert a[], int first, int last)
{
	int pivotElement;

	if (first < last)
	{
		pivotElement = pivot(a, first, last);
		quickSort(a, first, pivotElement - 1);
		quickSort(a, pivotElement + 1, last);
	}
}

/**
* Find and return the index of pivot element.
* @param a - The array.
* @param first - The start of the sequence.
* @param last - The end of the sequence.
* @return - the pivot element
*/
int AlertList::pivot(Alert a[], int first, int last)
{
	int  p = first;
	int pivotElement = a[first].getfips().fipsToInt();
	for (int i = first + 1; i <= last; i++)
	{
		if (a[i].getfips().fipsToInt() <= pivotElement)
		{
			p++;
			swap(a[i], a[p]);
		}
	}

	swap(a[p], a[first]);

	return p;
}
```

```
/**
* Swap the parameters.
* @param a - The first parameter.
* @param b - The second parameter.
*/
void AlertList::swap(Alert& a, Alert& b)
{
        Alert temp = a;
        a = b;
        b = temp;
}



void AlertList::findFips(string FIPS)
{
        Alert t;
        t.searchFipsCode(FIPS);
        cout << "county: " << t.getfips().getCounty() << endl
                << "FIPS code: " << t.getfips().getFipsCode() << endl
                << "State: " << t.getfips().getState() << endl
                << "Population: " << t.getPop() << endl;
}
```

```
------------------dateTime.cpp---------------

#include "dateTime.h";
#include <iostream>
#include <string>

        /*******************************************//**
        * **NOTE**
        * everything that has been clearly defined
        *through comments in the header will not be
        *redifined here. but some functions will have
        *a little extra if they do something either
        *hard to follow, or just need to clear them
        *up
        ***********************************************/

int dateTime::getMonth()
{
        return month;
}
int dateTime::getDay()
{
        return day;
}
int dateTime::getYear()
{
        return year;
}
int dateTime::getHr()
{
        return hr;
}
int dateTime::getMin()
{
        return min;
}

string dateTime::getTimeP()
{
        parseTime();
        return timeP;
}
string dateTime::getMonthP()
{
        parseMonth();
        return monthP;
}
void dateTime::setMonth(int toSet)
{
        month = toSet;
}
void dateTime::setDay(int toSet)
{
	day = toSet;
}
void dateTime::SetYear(int toSet)
{
	year = toSet;
}
void dateTime::setHr(int toSet)
{
	hr = toSet;
}
void dateTime::setMin(int toSet)
{
	min = toSet;
}
	/*******************************************//**
	*month var goes into a switch, and when apporaprate
	*case is found monthP is set to the new worded month
	*********************************************/
void dateTime::parseMonth()
{
	switch (month)
	{
	case 1:
		monthP = "January ";
		break;
	case 2:
		monthP = "February";
		break;
	case 3:
		monthP = "March";
		break;
	case 4:
		monthP = "April";
		break;
	case 5:
		monthP = "May";
		break;
	case 6:
		monthP = "June";
		break;
	case 7:
		monthP = "July";
		break;
	case 8:
		monthP = "August";
		break;
	case 9:
		monthP = "September";
		break;
	case 10:
		monthP = "October";
		break;
	case 11:
		monthP = "November";
		break;
	case 12:
		monthP = "December";
		break;
		default:
			break;
		}
}


		/*******************************************//**
		*concatanates the parsed hour, ":" , minutes, and
		*am or pm. if mins are 0, addes one zero to the mins
		*string changing it from 0 to 00. to convert form
		military time to normal, i just minus 12.
		**********************************************/
void dateTime::parseTime()
{
		string temp;
		if (hr <= 12)
		{
			if (min == 0)
			{
				temp = to_string(min) + "0";
			}
			if (hr == 0)
			{
				timeP = to_string(12) + ":" + temp + " AM";
			}
			else
			{
				timeP = to_string(hr) + ":" + temp + " AM";
			}
		}
		else
		{
			if (min == 0)
			{
				temp = to_string(min) + "0";
				timeP = to_string((hr - 12)) + ":" + temp + " PM";
			}
			else
			{
				timeP = to_string((hr - 12)) + ":" + to_string(min) + " PM";
			}
		}
}
```

```
---------------fips.cpp--------------

#include <string>
#include <sstream>
#include "fips.h"
using namespace std;

        /********************************************//**
        * **NOTE**
        * everything that has been clearly defined
        *through comments in the header will not be
        *redifined here. but some functions will have
        *a little extra if they do something either
        *hard to follow, or just need to clear them
        *up
        ***********************************************/

string fips::getFipsCode()
{
        return fipsCode;
}
string fips::getCounty()
{
        return county;
}
string fips::getState()
{
        return state;
}

void fips::setFipsCode(string toSet)
{
        fipsCode = toSet;
}
void fips::setCounty(string toSet)
{
        county = toSet;
}
void fips::setState(string toSet)
{
        state = toSet;
}
        /********************************************//**
        *uses a string stream to change from
        *an fipscode to an int. (string to int)
        ***********************************************/
int fips::fipsToInt()
{
        istringstream ss(fipsCode);
        int i;
```

```
		ss >> i;
		return i;
}
```

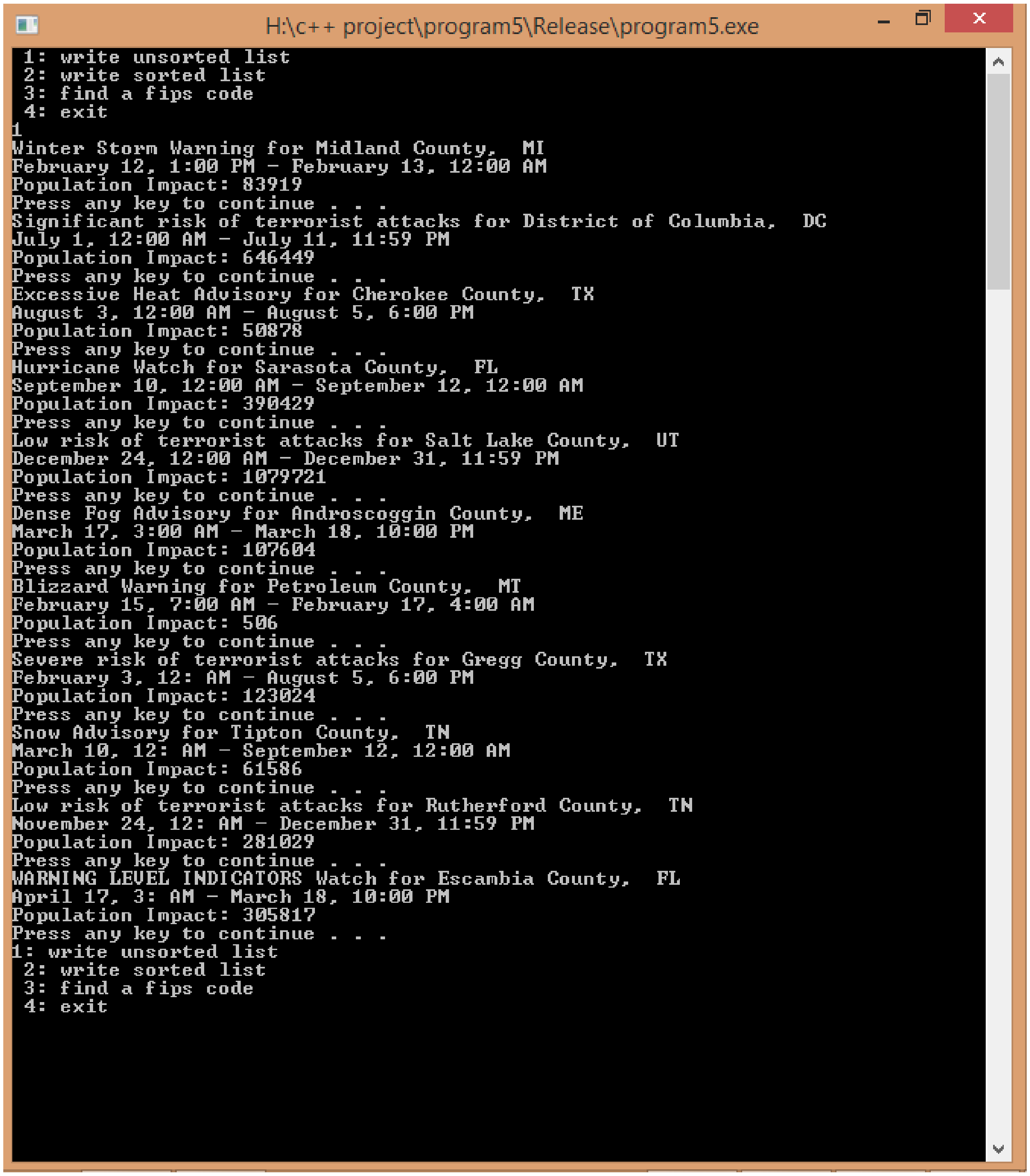
--------------------main.cpp----------------

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include "AlertList.h"
using namespace std;
/*******************************************//**
*\class main
*
* fips brief description.
*	this is the main driver
*
*drives the program witch user input, and apporate function calls
***********************************************/
int main() {
		AlertList al; //!< make the alertlist
		al.popList();
		int intUserIO; //!< user input
		string stringUserIO;  //!< fips code user input
		cout << " 1: write unsorted list \n 2: write sorted list"
				<< "\n 3: find a fips code \n 4: exit " << endl;
		cin >> intUserIO;
		while (intUserIO != 4)
		{

				if (intUserIO == 1)
				{
						al.writeList();
				}
				else if (intUserIO == 2)
				{
						al.writeSortedListByFips();
				}
				else
				{
						cout << "find what fips?" << endl;
						cin >> stringUserIO;
						al.findFips(stringUserIO);
				}
				cout << "1: write unsorted list \n 2: write sorted list"
						<< "\n 3: find a fips code \n 4: exit " << endl;
				cin >> intUserIO;
		}

		system("pause");
		return 0;
}
```

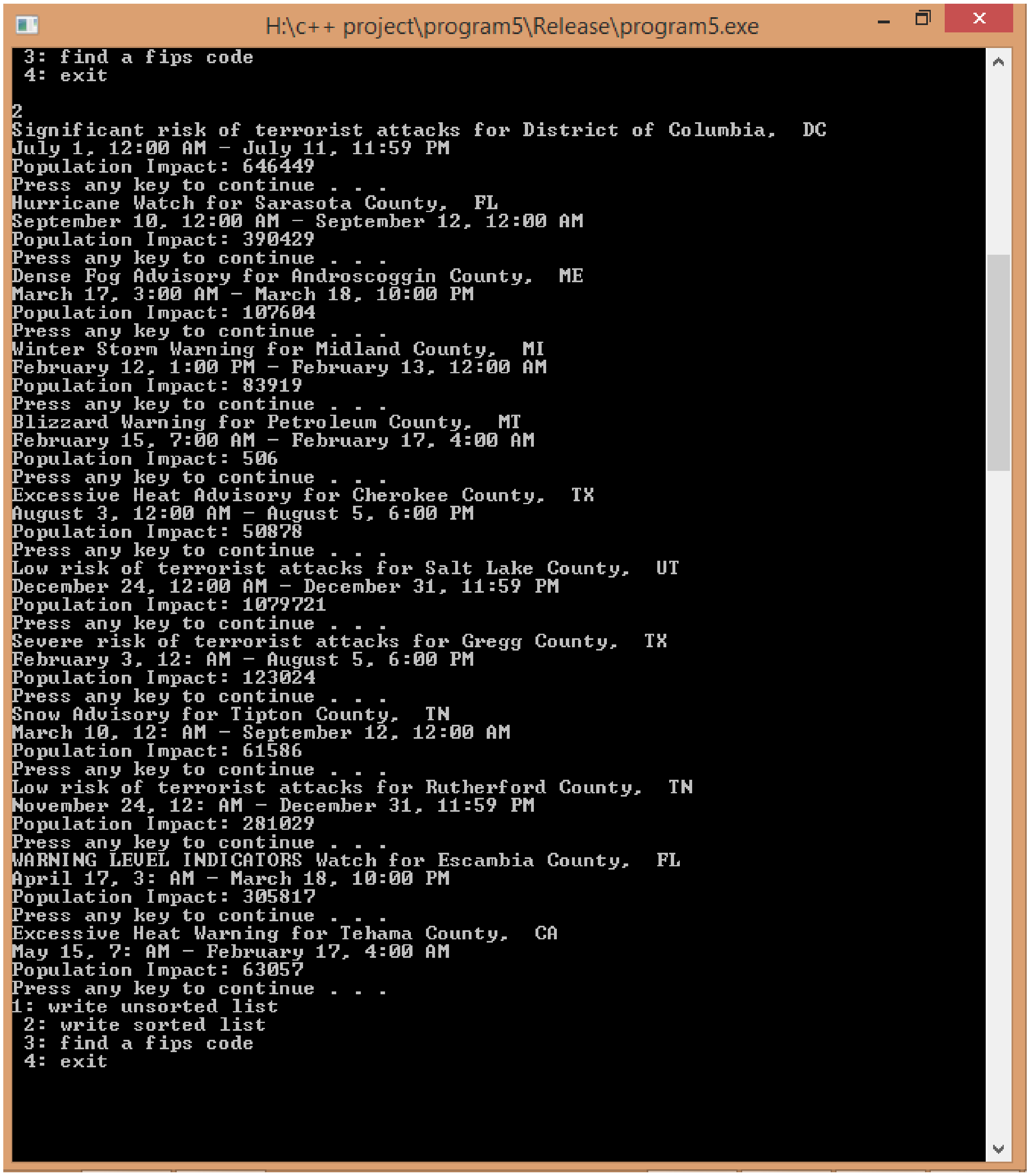
Write the unsorted list



H:\c++ project\program5\Release\program5.exe

```
 1: write unsorted list
 2: write sorted list
 3: find a fips code
 4: exit
1
Winter Storm Warning for Midland County,  MI
February 12, 1:00 PM - February 13, 12:00 AM
Population Impact: 83919
Press any key to continue . . .
Significant risk of terrorist attacks for District of Columbia,  DC
July 1, 12:00 AM - July 11, 11:59 PM
Population Impact: 646449
Press any key to continue . . .
Excessive Heat Advisory for Cherokee County,  TX
August 3, 12:00 AM - August 5, 6:00 PM
Population Impact: 50878
Press any key to continue . . .
Hurricane Watch for Sarasota County,  FL
September 10, 12:00 AM - September 12, 12:00 AM
Population Impact: 390429
Press any key to continue . . .
Low risk of terrorist attacks for Salt Lake County,  UT
December 24, 12:00 AM - December 31, 11:59 PM
Population Impact: 1079721
Press any key to continue . . .
Dense Fog Advisory for Androscoggin County,  ME
March 17, 3:00 AM - March 18, 10:00 PM
Population Impact: 107604
Press any key to continue . . .
Blizzard Warning for Petroleum County,  MT
February 15, 7:00 AM - February 17, 4:00 AM
Population Impact: 506
Press any key to continue . . .
Severe risk of terrorist attacks for Gregg County,  TX
February 3, 12: AM - August 5, 6:00 PM
Population Impact: 123024
Press any key to continue . . .
Snow Advisory for Tipton County,  TN
March 10, 12: AM - September 12, 12:00 AM
Population Impact: 61586
Press any key to continue . . .
Low risk of terrorist attacks for Rutherford County,  TN
November 24, 12: AM - December 31, 11:59 PM
Population Impact: 281029
Press any key to continue . . .
WARNING LEVEL INDICATORS Watch for Escambia County,  FL
April 17, 3: AM - March 18, 10:00 PM
Population Impact: 305817
Press any key to continue . . .
1: write unsorted list
 2: write sorted list
 3: find a fips code
 4: exit
```

Write the sorted list

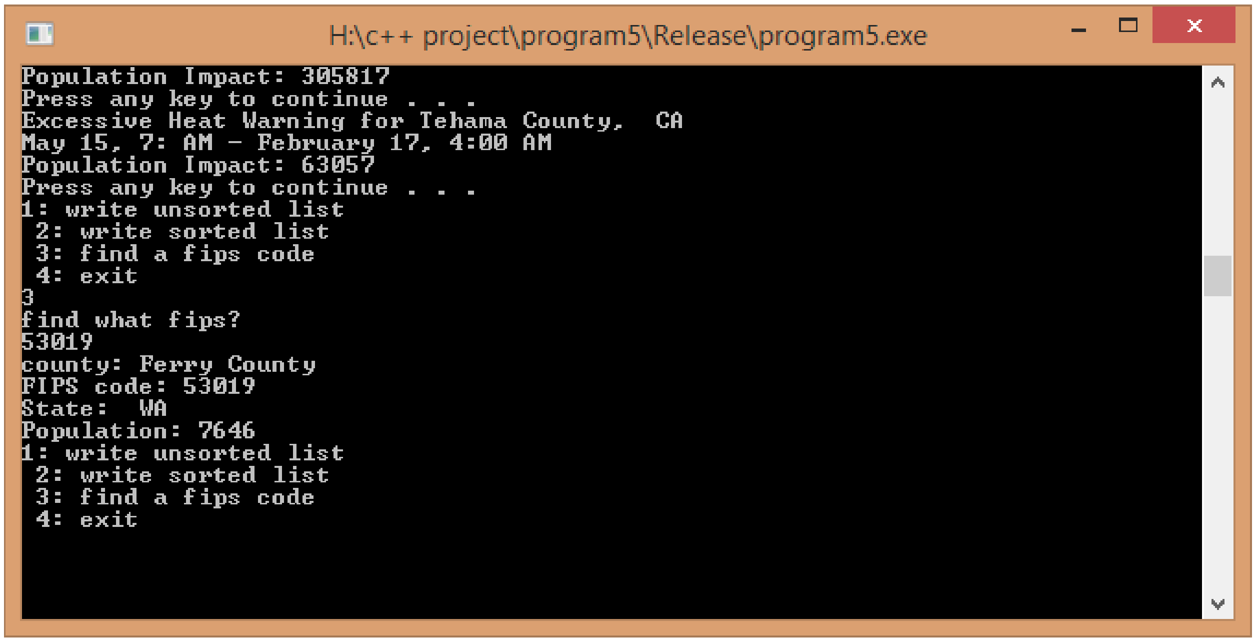


H:\c++ project\program5\Release\program5.exe

```
 3: find a fips code
 4: exit

2
Significant risk of terrorist attacks for District of Columbia,  DC
July 1, 12:00 AM - July 11, 11:59 PM
Population Impact: 646449
Press any key to continue . . .
Hurricane Watch for Sarasota County,  FL
September 10, 12:00 AM - September 12, 12:00 AM
Population Impact: 390429
Press any key to continue . . .
Dense Fog Advisory for Androscoggin County,  ME
March 17, 3:00 AM - March 18, 10:00 PM
Population Impact: 107604
Press any key to continue . . .
Winter Storm Warning for Midland County,  MI
February 12, 1:00 PM - February 13, 12:00 AM
Population Impact: 83919
Press any key to continue . . .
Blizzard Warning for Petroleum County,  MT
February 15, 7:00 AM - February 17, 4:00 AM
Population Impact: 506
Press any key to continue . . .
Excessive Heat Advisory for Cherokee County,  TX
August 3, 12:00 AM - August 5, 6:00 PM
Population Impact: 50878
Press any key to continue . . .
Low risk of terrorist attacks for Salt Lake County,  UT
December 24, 12:00 AM - December 31, 11:59 PM
Population Impact: 1079721
Press any key to continue . . .
Severe risk of terrorist attacks for Gregg County,  TX
February 3, 12: AM - August 5, 6:00 PM
Population Impact: 123024
Press any key to continue . . .
Snow Advisory for Tipton County,  TN
March 10, 12: AM - September 12, 12:00 AM
Population Impact: 61586
Press any key to continue . . .
Low risk of terrorist attacks for Rutherford County,  TN
November 24, 12: AM - December 31, 11:59 PM
Population Impact: 281029
Press any key to continue . . .
WARNING LEVEL INDICATORS Watch for Escambia County,  FL
April 17, 3: AM - March 18, 10:00 PM
Population Impact: 305817
Press any key to continue . . .
Excessive Heat Warning for Tehama County,  CA
May 15, 7: AM - February 17, 4:00 AM
Population Impact: 63057
Press any key to continue . . .
1: write unsorted list
 2: write sorted list
 3: find a fips code
 4: exit
```

Find a county by fips code



```
Population Impact: 305817
Press any key to continue . . .
Excessive Heat Warning for Tehama County,  CA
May 15, 7: AM - February 17, 4:00 AM
Population Impact: 63057
Press any key to continue . . .
1: write unsorted list
 2: write sorted list
 3: find a fips code
 4: exit
3
find what fips?
53019
county: Ferry County
FIPS code: 53019
State:  WA
Population: 7646
1: write unsorted list
 2: write sorted list
 3: find a fips code
 4: exit
```