```
/*
 * myDate.cpp
 *
 *  Created on: Sep 18, 2016
 *      Author: Kenny Do
 */

#include "myDate.h"
#include <stdlib.h>
#include <string.h>
#include <iomanip>
#include <iostream>
#include <cstdlib>
#include <time.h>

using namespace std;

/**
 * Default contructor
 */
myDate::myDate() {
    defaultDate();
}

/**
 * Sets the default date
 */
void myDate::defaultDate() {
    month = 5;
    day = 11;
    year = 1959;
}

/**
 * Constructor that takes in the month, day, and year
 */
myDate::myDate(int M, int D, int Y) {
    if(M > 12 || M < 1 || D < 1) {
        defaultDate();
    } else {
        month = M;
        day = D;
        year = Y;
        switch(month) {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            if(D > 31) {
                defaultDate();
            }
            break;
        case 2:
            if((Y % 4 == 0 && D > 29) || (Y % 4 != 0 && D > 28)) {
                defaultDate();
            }
            break;
        case 4:
        case 6:
        case 9:
        case 11:
            if(D > 30) {
                defaultDate();
```

```
            }
            break;

        }
    }
}

/**
 * Displays the date
 */
void myDate::display() {
    switch(month) {
    case 1:
        cout << "January";
        break;
    case 2:
        cout << "February";
        break;
    case 3:
        cout << "March";
        break;
    case 4:
        cout << "April";
        break;
    case 5:
        cout << "May";
        break;
    case 6:
        cout << "June";
        break;
    case 7:
        cout << "July";
        break;
    case 8:
        cout << "August";
        break;
    case 9:
        cout << "September";
        break;
    case 10:
        cout << "October";
        break;
    case 11:
        cout << "November";
        break;
    case 12:
        cout << "December";
        break;
    }
    cout << " " << day << ", " << year << setw(10);
}


/**
 * Increment the date by N
 */
void myDate::incrDate(int N) {
    int julian = getJulianDate() + N;
    myDate newCalendar = returnGregorian(julian);
    month = newCalendar.getMonth();
    day = newCalendar.getDay();
    year = newCalendar.getYear();
}

/**
 * Decrement the date by N
```

```
 */
void myDate::decrDate(int N) {
    int julian = getJulianDate() - N;
    myDate newCalendar = returnGregorian(julian);
    month = newCalendar.getMonth();
    day = newCalendar.getDay();
    year = newCalendar.getYear();
}

/**
 * Calculates the days between D and this date
 */
int myDate::daysBetween(myDate D) {
    return D.getJulianDate() - getJulianDate();
}

/**
 * Returns the month
 */
int myDate::getMonth() const {
    return month;
}

/**
 * Returns the day
 */
int myDate::getDay() const {
    return day;
}

/**
 * Returns the year
 */
int myDate::getYear() const {
    return year;
}

/**
 * Returns the days between this date and the start of the year
 */
int myDate::getYearOffset() const {
    myDate start(1, 1, year);
    return getJulianDate() - start.getJulianDate();
}

/**
 * Returns the Julian number of this date
 */
double myDate::getJulianDate() const {
    return day - 32075 + 1461 * (year + 4800 + (month - 14) / 12) / 4 +367 * (month - 2 -
    (month - 14) / 12 * 12) /12 - 3 * ((year +4900 + (month - 14) / 12) / 100) / 4;
}

/**
 * Returns a myDate object that is calculated from the julian number
 */
myDate myDate::returnGregorian(int julian) {
    const int temp11 = julian + 68569;
    const int temp21 = 4 * temp11 / 146097;
    const int temp12 = temp11 - (146097 * temp21 + 3) / 4;
    const int year1 = 4000 * (temp12 + 1) / 1461001;
    const int temp13 = temp12 - (1461 * year1 / 4) + 31;
    const int month1 = 80 * temp13 / 2447;
    const int day1 = temp13 - (2447 * month1 / 80);
    const int temp14 = month1 / 11;
    const int month2 = month1 + 2 - 12 * temp14;
```

```
    const int year2 = 100 * (temp21 - 49) + year1 + temp14;

    return myDate(month2, day1, year2);
}

/**
 * Returns the display as a cstring
 */
const char* myDate::toString() {
    char * cString = new char[50];
    switch(month) {
    case 1:
        strcpy(cString, "January");
        break;
    case 2:
        strcpy(cString, "February");
        break;
    case 3:
        strcpy(cString, "March");
        break;
    case 4:
        strcpy(cString, "April");
        break;
    case 5:
        strcpy(cString, "May");
        break;
    case 6:
        strcpy(cString, "June");
        break;
    case 7:
        strcpy(cString, "July");
        break;
    case 8:
        strcpy(cString, "August");
        break;
    case 9:
        strcpy(cString, "September");
        break;
    case 10:
        strcpy(cString, "October");
        break;
    case 11:
        strcpy(cString, "November");
        break;
    case 12:
        strcpy(cString, "December");
        break;
    }
    strcat(cString, " ");
    char dateDay[20];
    itoa (day,dateDay,10);
    strcat(cString, dateDay);

    strcat(cString, ", ");
    char dateYear[200];
    itoa (year,dateYear,10);
    strcat(cString, dateYear);
    return cString;

}

/**
 * Returns true if the this date is greater than to the other date else false
 */
bool myDate::operator >(const myDate & date) {
    if(this->daysBetween(date) < 0) {
```

```
        return true;
    } else {
        return false;
    }
}

/**
 * Returns true if the this date is less than to the other date else false
 */
bool myDate::operator <(const myDate& date) {
    if(this->daysBetween(date) > 0) {
        return true;
    } else {
        return false;
    }
}

/**
 * Returns true if the this date is greater than or equal to the other date else false
 */
bool myDate::operator >=(const myDate& date) {
    if(this->daysBetween(date) <= 0) {
        return true;
    } else {
        return false;
    }
}

/**
 * Returns true if the this date is less than or equal to the other date else false
 */
bool myDate::operator <=(const myDate& date) {
    if(this->daysBetween(date) >= 0) {
        return true;
    } else {
        return false;
    }
}

/**
 * Returns true if this date is equal to the other date
 */
bool myDate::operator ==(const myDate& date) {
    if(this->daysBetween(date) == 0) {
        return true;
    } else {
        return false;
    }
}

/**
 * Returns true if this date is not equal to the other date
 */
bool myDate::operator !=(const myDate& date) {
    if(this->daysBetween(date) != 0) {
        return true;
    } else {
        return false;
    }
}

/**
 * Returns a day between d1 and d2
 */
myDate myDate::getRandomDayBetween(myDate d1, myDate d2) {
    int d1Julian = d1.getJulianDate();
```

```
    int d2Julian = d2.getJulianDate();
    int randJulian = rand() % (d2Julian - d1Julian) + d1Julian;
    return returnGregorian(randJulian);
}
```