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

#include "upDate.h"

using namespace std;

/**
 * Initialize numOfDateObjects to 0
 */
int upDate::numOfDateObjects = 0;

/**
 * Creates a upDate with the default
 */
upDate::upDate() {
    iptr = new int[3];
    ++numOfDateObjects;
    defaultDate();
}

/**
 * Creates a upDate with M, D, Y
 */
upDate::upDate(int M, int D, int Y) {
    iptr = new int[3];
    ++numOfDateObjects;
    setDate(M, D, Y);
}

/**
 * Creates a clone of date
 */
upDate::upDate(const upDate& date) {
    iptr = new int[3];
    ++numOfDateObjects;
    iptr[0] = date.getMonth();
    iptr[1] = date.getDay();
    iptr[2] = date.getYear();
}

/**
 * Destructor for update
 */
upDate::~upDate() {
    delete[] iptr;
    numOfDateObjects--;
//  iptr = nullptr;
}

/**
 * Sets the date to May 11, 1959
 */
```

```
void upDate::defaultDate() {
    iptr[0] = 5;
    iptr[1] = 11;
    iptr[2] = 1959;
}

/**
 * Sets the date to M, D, Y for month, day, year respectively
 */
void upDate::setDate(int M, int D, int Y) {
    iptr[0] = M;
    iptr[1] = D;
    iptr[2] = Y;
    int julianFromConstructor = julian();
    updateFromGregorian(julianFromConstructor);
    if(M == iptr[0] && D == iptr[1] && Y == iptr[2] ) {
        iptr[0] = M;
        iptr[1] = D;
        iptr[2] = Y;
    } else {
        defaultDate();
    }

}

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

/**
 * Returns the month
 */
int upDate::getMonth() const {
    return iptr[0];
}

/**
 * Returns the toString of the month
 */
std::string upDate::getMonthName() {
    switch(iptr[0]) {
    case 1:
        return "January";
        break;
    case 2:
        return "February";
        break;
    case 3:
        return "March";
        break;
    case 4:
        return "April";
        break;
    case 5:
```

```
        return "May";
        break;
    case 6:
        return "June";
        break;
    case 7:
        return "July";
        break;
    case 8:
        return "August";
        break;
    case 9:
        return "September";
        break;
    case 10:
        return "October";
        break;
    case 11:
        return "November";
        break;
    case 12:
        return "December";
        break;
    }
    return "";
}

/**
 * Returns the day
 */
int upDate::getDay() const {
    return iptr[1];
}

/**
 * Returns the year
 */
int upDate::getYear() const {
    return iptr[2];
}

/**
 * Return the amount of upDate objects in the program
 */
int upDate::GetDateCount() {
    return numOfDateObjects;
}

/**
 * Returns the Julian number of this date
 */
double upDate::julian() const {
    return iptr[1] - 32075 + 1461 * (iptr[2] + 4800 + (iptr[0] - 14) / 12) / 4 +367 * (iptr[0] -
2 - (iptr[0] - 14) / 12 * 12) /12 - 3 * ((iptr[2] +4900 + (iptr[0] - 14) / 12) / 100) / 4;
}

/**
```

```
 * Returns a upDate object that is calculated from the julian number
 */
upDate upDate::returnGregorian(int julian) {
    int year, month, day;
    int L = julian + 68569;
    int N = 4 * L / 146097;
    L = L - (146097 * N + 3) / 4;
    year = 4000 * (L + 1) / 1461001;
    L = L - 1461 * year / 4 + 31;
    month = 80 * L / 2447;
    day = L - 2447 * month / 80;
    L = month / 11;
    month = month + 2 - 12 * L;
    year = 100 * (N - 49) + year + L;

    return upDate(month, day, year);
}

/**
 * Update this upDate object with the julian
 */
void upDate::updateFromGregorian(int julian) {
    int year, month, day;
    int L = julian + 68569;
    int N = 4 * L / 146097;
    L = L - (146097 * N + 3) / 4;
    year = 4000 * (L + 1) / 1461001;
    L = L - 1461 * year / 4 + 31;
    month = 80 * L / 2447;
    day = L - 2447 * month / 80;
    L = month / 11;
    month = month + 2 - 12 * L;
    year = 100 * (N - 49) + year + L;

    iptr[0] = month;
    iptr[1] = day;
    iptr[2] = year;
}

/**
 * Overloads the << with the toString of this class
 */
std::ostream& operator<<(std::ostream& os, const upDate& date) {
    os << date.iptr[0] <<"/" << date.iptr[1]<<"/"<<date.iptr[2];
    return os;
}

/**
 * Assign this upDate to date
 */
void upDate::operator =(const upDate & date) {
    iptr[0] = date.getMonth();
    iptr[1] = date.getDay();
    iptr[2] = date.getYear();
}
/**
 * Increment the date by N
```

```
 */
upDate upDate::operator +(int N) {
    int jul = julian() + N;
    return returnGregorian(jul);
}

/**
 * Post-Increment the date by 1
 */
upDate& upDate::operator ++() {
    int jul = this->julian() + 1;
    updateFromGregorian(jul);
    return *this;
}

/**
 * Pre-Increment the date by 1
 */
upDate upDate::operator ++(int int1) {
    upDate temp = *this;
    ++*this;
    return temp;
}

/**
 * Post-Decrement the date by 1
 */
upDate& upDate::operator --() {
    int jul = this->julian() - 1;
    updateFromGregorian(jul);
    return *this;
}

/**
 * Pre-Decrement the date by 1
 */
upDate upDate::operator --(int int1) {
    upDate temp = *this;
    --*this;
    return temp;
}

/**
 * Adds a int + upDate object
 */
upDate operator +(int N,const upDate& date) {
    int jul = date.julian() + N;
    return date.returnGregorian(jul);
}

/**
 * Decrement the date by N
 */
upDate upDate::operator -(int N) {
    int jul = julian() - N;
    return returnGregorian(jul);
}
```

```
/**
 * Subtract this date with another date
 */
int upDate::operator -(const upDate& date) {
    return this->julian() - date.julian();
}

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

/**
 * Returns true if this date is equal to the other date
 */
bool upDate::operator ==(const upDate& date) {
    if(this->daysBetween(date) == 0) {
```

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

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