

CSCI 135 Software Design and Analysis I

Homework 5

Solution

Saad Mneimneh
Visiting Professor
Hunter College of CUNY

Problem 1: Date class

Define a class called Date to represent the concept of a date. Your class should have three integers as private member data representing the month, the day, and the year. You should also provide the following:

- A constructor that accepts three integers and sets the month, day, and year accordingly.
- A constructor that accepts two integers and sets the month and the day, and sets the year to the current year.
- A default constructor that sets the month, day, and year to the current date.
- A member function that accepts three integers and sets the month, day, and year accordingly (this may be used by all constructors).
- Appropriate member functions to return information about the date, i.e. the month, the day, and the year.

The constructors and the set function must perform appropriate checking. Here's a suggested piece of code that you can use/modify to help you check if a given date is valid:

```
bool leap(int y) {
    //you do it
}

int lastday(int m, int y) {
    int mdays[12]={31,28,31,30,31,30,31,31,30,31,30,31};
    if (leap(y))
        mdays[1]=29; //feb
    return mdays[m-1];
}
```

```

bool valid(int m, int d, int y) {
    if (m<1 || m>12 || y<1 || d<1 || d>lastday(m,y))
        return false;
    else
        return true;
}

```

The following program shows an example of how to get information about the current date (you will need this for the constructors, especially the default constructor).

```

#include <ctime>

int main() {
    time_t t=time(0);
    tm * p=localtime(&t);
    cout<<p->tm_mon+1<<'/'>> //that's the current month
    cout<<p->tm_mday<<'/'>> //that's the current day
    cout<<p->tm_year+1900<<'\\n'>> //that's the current year
}

```

ANSWER: Here's the class declaration which can be saved as date.h:

```

#include <iostream>

using std::ostream;

class Date {
    int m;
    int d;
    int y;

public:
    Date();
    Date(int, int, int);
    Date(int, int);

    void set(int, int, int, bool check=true);

    int month()const {
        return m;
    }

    int day()const {
        return d;
    }

    int year()const {
        return y;
    }
};

```

Here's the class definition which can be saved as date.c:

```
#include "date.h"
#include <ctime>

using std::cout;

//some helper functions
bool leap(int y) {
    return (y%400==0 || (y%4==0 && y%100!=0));
}

int lastday(int m, int y) {
    int mdays[12]={31,28,31,30,31,30,31,31,30,31,30,31};
    if (leap(y))
        mdays[1]=29; //feb
    return mdays[m-1];
}

bool valid(int m, int d, int y) {
    if (m<1 || m>12 || y<1 || d<1 || d>lastday(m,y)) //short circuit evaluation
        return false;
    else
        return true;
}

Date::Date() {
    time_t t=time(0);
    tm * p=localtime(&t);
    set(p->tm_mon+1,p->tm_mday,p->tm_year+1900,false);
}

Date::Date(int m, int d, int y) {
    set(m,d,y);
}

Date::Date(int m, int d) {
    time_t t=time(0);
    tm * p=localtime(&t);
    set(m,d,p->tm_year+1900);
}
```

```

void Date::set(int m, int d, int y, bool check) {
    this->m=m;
    this->d=d;
    this->y=y;
    if (check && !valid(m,d,y)) { //short circuit evaluation
        time_t t=time(0);
        tm * p=localtime(&t);
        this->m=p->tm_mon+1;
        this->d=p->tm_mday;
        this->y=p->tm_year+1900;
    }
}

```

Problem 2: Operator overloading

Provide the following operator overloading functions as member functions of the Date class:

- operator ++ advances the date by one day (pre and post)
- operator -- retracts the date by one day (pre and post)
- operator - takes two dates and returns their difference in number of days

Provide the following operator overloading functions as functions outside the Date class:

- operator + takes a date d and an integer n and returns a date corresponding to date d plus n days
- operator - takes a date d and an integer n and returns a date corresponding to date d minus n days
- operator << takes an output stream and a date and outputs the date in the format m/d/y

The following must work:

```

int main() {
    Date d1=Date(1,13,1973);
    Date d2;
    d2=d1+3;
    d2=3+d1;
    d2=d1-3;
    d2=d1+3;
    cout<<d1<<d2;
}

```

PS: Use pass by reference and const as much as you can.

ANSWER: Here's the class Date with the operators added:

```

class Date {
    . . .

```

```

const Date operator++();
const Date operator++(int);
const Date operator--();
const Date operator--(int);

int operator-(const Date&);
}

```

Here's the implementation in date.c (also showing operators defined outside the class)

```

//pre increment
const Date Date::operator++() {
    d++;
    if (!valid(m,d,y)) {
        d=1;
        m++;
        if (!valid(m,d,y)) {
            m=1;
            y++;
        }
    }
    return *this;
}

//post increment
const Date Date::operator++(int) {
    Date temp=*this;
    ++(*this); //pre
    return temp;
}

//pre decrement
const Date Date::operator--() {
    d--;
    if (!valid(m,d,y)) {
        m--;
        d=lastday(m,y);
        if (!valid(m,d,y)) {
            d=31;
            m=12;
            y--;
        }
    }
    return *this;
}

```

```

//post decrement
const Date Date::operator--(int) {
    Date temp=*this;
    --(*this); //pre
    return temp;
}

int Date::operator-(const Date& da) {
    Date start=*this;
    Date end=da;
    if (!(y<da.y ||
        (y==da.y && m<da.m) ||
        (y==da.y && m==da.m && d<da.d))) {
        start=da;
        end=*this;
    }
    int days=0;
    while (!(start.m==end.m && start.d==end.d && start.y==end.y)) {
        ++days;
        ++start;
    }
    return days;
}

//need to be declared because used by operator+
const Date operator-(const Date&d, int n);

const Date operator+(const Date& d, int n) {
    if (n<0)
        return d-(-n);//call operator-
    Date result=d;
    for (int i=0; i<n; i++)
        ++result;
    return result;
}

const Date operator+(int n, const Date& d) {
    return d+n;
}

const Date operator-(const Date& d, int n) {
    if (n<0)
        return d+(-n);//call operator+
    Date result=d;
    for (int i=0; i<n; i++)
        --result;
    return result;
}

```

```
ostream& operator<<(ostream& s, const Date& d) {
    s<<d.month()<<"/"<<d.day()<<"/"<<d.year();
    return s;
}
```

Problem 3: String reversal

Write a function that reverses a string:

```
void reverse(char * t, int n, const char * s) {...}
```

The string *s* is the original string. The string *t* of length *n* is the one that will hold *s* reversed. Make sure to check that the length of *t* is enough to hold *s* reversed. If not, *t* will hold as many characters as it can fit.

ANSWER: We are going to assume that *t* is long enough so we are not going to check for anything.

```
int min(int i, int j) { if (i<j) return i; else return j;

void reverse(char * t, int n, const char * s) {
    int len=strlen(s);
    int true_len=min(len,n);
    for (int i=0; i<true_len; i++)
        t[i]=s[len-1-i];
    t[true_len]='\0';
}
```

Problem 4: String censoring

Write a function that will censor a string using a dictionary of bad words.

```
void censor(char * s, const char * dict[], int n) {...}
```

In the above function, *dict* is an *n*-element array of strings. Here's an example of how the function can be used:

```
int main() {
    char s[]='fwording shit! what the fword is happening?';
    const char * dict[2]={'shit', 'fword'};
    censor(s,dict,2);
    cout<<s;
}
```

The output should be: *****ing *****! what the ***** is happening?

ANSWER:

```
void censor(char * s, const char * dict[], int n) {
    for (int i=0; i<n; i++) {
        char * word=dict[i];
        int len=strlen(word);
        for (int j=0; j<=strlen(s)-len; j++) {
            bool match=true;
            for (int k=0; k<len; k++)
                match=match && (s[j+k]==word[k]);
        }
    }
}
```

```

        if (match)
            for (int k=0; k<len; k++)
                s[j+k]='*';
    }
}

```

Problem 5: Polygon

Consider the following simple Point class:

```

class Point {
public:
    float x;
    float y;
}

```

Define a class Polygon that consists of a number of Points.

```

class Polygon {
    Point * points; //the points
    int size; //number of points

public:
    Polygon() {...};
    Polygon(const Point * pts, int n) {...};

    Point getPoint(int n)const {
        //returns the nth point
    }

    void setPoint(const Point& p, int n) {
        //sets the nth point
    }

    int getSize()const {
        return size;
    }
    . . .
};

```

(a) Complete the class and declare the destructor, the copy constructor, and the assignment operator. Assume that the default constructor constructs an empty polygon.

ANSWER:

```

class Polygon {
    Point * points; //the points
    int size; //number of points
public:
    Polygon() {
        points=0; //NULL pointer
        size=0;
    }
}

```

```

Polygon(const Point * pts, int n) {
    size=n;
    points=new Point[size];
    for (int i=0; i<size; i++)
        points[i]=pts[i];
}

Point getPoint(int n)const {
    return points[n];
}

void setPoint(const Point& p, int n) {
    points[n]=p;
}

int getSize()const {
    return size;
}

Polygon(const Polygon& poly) { //copy constructor
    size=poly.size;
    points=new Point[size];
    for (int i=0; i<size; i++)
        points[i]=poly.points[i];
}

Polygon& operator=(const Polygon& poly) { //assignment operator
    if (this==&poly)
        return *this;
    delete[] points;
    size=poly.size;
    points=new Point[size];
    for (int i=0; i<size; i++)
        points[i]=poly.points[i];
    return *this;
}

~Polygon() { //destructor
    delete[] points;
    points=0; //null the pointer
}
};

```

(b) Change the class Point to have both of its members private, and add a constructor Point(int, int) to construct a point given the two parameters x and y . By doing so, the Polygon class will be affected and will not compile. Discover why and fix the problem.

```
class Point {
private:
    float x;
    float y;

public:
    Point(float x, float y) {
        this->x=x;
        this->y=y;
    }
};
```

The problem now is that Point lost its default constructor. The polygon class cannot create dynamically an array of Points anymore, because it needs to call the default constructor of Point to do that. To fix the problem, we restore a default constructor.

```
class Point {
private:
    float x;
    float y;

public:
    Point(float x, float y) {
        this->x=x;
        this->y=y;
    }

    Point() {
        x=0;
        y=0;
    }
};
```