МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Лабораторная работа №3

по дисциплине

«Информатика и программирование»

Модель календаря

Группа: **АБс-822**

Студент: **Соленый Н.Е.**

Преподаватель:

НОВОСИБИРСК

2019

**Задача:**

Написать программу реализующую работу с упрощенным григорианским календарем (датами).

Использовать следующие правила для календаря:

· Минимальный год — 1-ый год н.э., максимальный год 9999 н.э.

· Момент времени представляется годом, месяцем, числом месяца, часом, минутами, секундами

· Каждый год, номер которого кратен 4 является високосным

· Каждый год, номер которого кратен 100 не является високосным

· Каждый год, номер которого кратен 400 является високосным

· Часовой пояс не учитывается (считается равным UTC)

· Не учитывать правило, согласно которому следующим днем после 4 октября 1582 года идет 15 октября 1582 года

· Не требуется совпадение юлианского и данного календаря в датах до 4 октября 1582 года (проблема високосных годов кратных 100 но не кратных 400)

Класс даты должен поддерживать корректные операции приращения минут, секунд, часов, дней, месяцев и лет с учетом правил календаря.

Месяцы реализовать перечислением Month {Jan=1, …, Dec = 12}.

Конструктор по умолчанию должен создавать дату соответствующую текущему времени UTC (для получения времени можно использовать функции из time.h)

Класс должен реализовывать конструктор копии.

Класс должен обеспечить полный набор методов селекторов для получения текущего года, месяца, дня месяца (числа), часа, минут, секунд.

Реализовать полный конструктор (unsigned аргументы — год, месяц, число, час, минуты, секунды).

В случае если значения в конструктор переданы некорректно, осуществлять автоматическую нормализацию даты (значение большее допустимого в определенном поле приводит к увеличению старших полей, таким образом вызов конструктора от аргументов Date(2010, Feb, 30, 22, 59, 72) приведет к созданию объекта соответствующего Date(2010, Mar, 2, 23, 00, 13))

Реализовать дополнительно конструкторы:

· Date (unsigned int year, Month m, unigned int day) — создает объект с временем 0 ч 0 м 0 сек

· Date (unsigned int hrs, unsigned int mnts, unsigned int days) - создает объект с указанным временем и текущей датой

Операции addYears, addMonths, addDays, addHours, addMinutes, addSeconds должны возвращать новый объект «по значению», оставляя исходный объект неизменным.

Класс должен реализовывать оператор присваивания (единственный метод-модификатор в классе).

Таким образом для изменения исходного объекта нужно пользоваться конструкцией вида:

Date theDate = theDate.addMonths(1);

Аргументы методов addXXX могут быть отрицательными, но при этом дата не может быть меньше 0:00:00 1 Jan 1 или больше 9999-12-31 23:59:59

Реализовать метод std::string toString() const возвращающий строковое представление даты в формате:

YYYY-MMM-DD hh::mm::ss

**Структура проекта**

// (main.cpp)

#include "class.hpp"

using namespace std;

int main() {

using namespace datetime;

Date date;

Date d = date.addYears(0).addMonths(0).addTime(5day+8hour+2min+4sec);

cout << date.toString() << endl;

cout << d.toString() << endl;

//cout << date.day << " " << date.hour << "h " << date.min << "min " << date.sec << "sec " << endl;

//cout << d.day << " " << d.hour << "h " << d.min << "min " << d.sec << "sec " << endl;

//cout << d.toString() << endl;

return 0;

}

// (class.hpp)

#include <iostream>

#include <limits>

#include <time.h>

namespace datetime {

int64\_t constexpr operator "" sec(uint64\_t sec) { if(sec>std::numeric\_limits<int64\_t>::max()) throw std::invalid\_argument("Compile error. Too big numeric constant"); return sec; }

int64\_t constexpr operator "" min(uint64\_t sec) { uint64\_t d = sec\*60; if(d>std::numeric\_limits<int64\_t>::max()) throw std::invalid\_argument("Compile error. Too big numeric constant"); return d; }

int64\_t constexpr operator "" hour(uint64\_t sec) { uint64\_t d = sec\*3600; if(d>std::numeric\_limits<int64\_t>::max()) throw std::invalid\_argument("Compile error. Too big numeric constant"); return d; }

int64\_t constexpr operator "" day(uint64\_t sec) { uint64\_t d = sec\*24\*3600; if(d>std::numeric\_limits<int64\_t>::max()) throw std::invalid\_argument("Compile error. Too big numeric constant"); return d; }

enum class Month { Jan=1, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };

enum class Weekday { Sun=1, Mon, Tue, Wed, Thu, Fri, Sat };

class Date {

private:

int32\_t year, day, hour, min, sec;

Month month;

Weekday weekday;

tm \*time;

void reflect(const Date& date);

public:

Date(const Date &date);

Date(); /// creates a current day with a current time

~Date();

Date(int32\_t year, Month m, uint32\_t day, uint32\_t seconds); /// creates a date at a time given in seconds

Date(int32\_t year, Month m, uint32\_t day); /// creates a day with 0h 0m 0s

Date addYears(int32\_t years);

Date addMonths(int32\_t months);

Date addTime(int32\_t seconds);

std::string toString();

};

}

// (class.cpp)

#include "class.hpp"

#include <malloc.h>

#include <string.h>

using namespace std;

using namespace datetime;

Date::Date() {

time\_t seconds = ::time(NULL);

tm \*t = gmtime(&seconds);

year = 1900+t->tm\_year;

month = static\_cast<Month>(1+t->tm\_mon);

day = t->tm\_mday + 1;

weekday = static\_cast<Weekday>(1+t->tm\_wday);

hour = t->tm\_hour;

min = t->tm\_min + (t->tm\_sec==60?1:0);

this->sec = (t->tm\_sec < 60 ? t->tm\_sec : 0);

time = t;

}

Date::Date(int32\_t year, Month m, uint32\_t day) {

this->year = year;

this->month = m;

this->day = day;

this->weekday = Weekday::Sun;

hour = min = sec = 0;

}

Date::Date(int32\_t year, Month m, uint32\_t day, uint32\_t seconds) {

sec = seconds%1min;

min = seconds/1min;

hour = seconds/1hour + min/60;

min%=1hour;

this->day = day + hour/24;

hour%=24;

int n = static\_cast<typename std::underlying\_type<Month>::type>(m);

int c = 0;

switch(m){ case Month::Jan: c = 31; break;

case Month::Feb: c = 28; if( (!(year%4) && !!(year%100) ) ||!(year%400))c = 29; break;

case Month::Mar: c = 31; break;

case Month::Apr: c = 30; break;

case Month::May: c = 31; break;

case Month::Jun: c = 30; break;

case Month::Jul: c = 31; break;

case Month::Aug: c = 31; break;

case Month::Sep: c = 30; break;

case Month::Oct: c = 31; break;

case Month::Nov: c = 30; break;

case Month::Dec: c = 31; break; }

n+=(this->day-1)/c;

this->year = year + (n-1)/12;

if (year < -1 || year > 9999) throw std::runtime\_error("Year is out of range!");

this->month = static\_cast<Month>((n-1)%12+1);

}

Date::~Date() {

if(time) free(time);

}

std::string Date::toString() {

//char \*msg = ::asctime(time); // do not free it!

//string str(msg);

//str[str.length()-1]=' ';

//return str+"(UTC)\n";

char msg[80] = {0};

sprintf(msg, "%d %s%d %s%d, %s%dh %s%dm %s%ds",

year, (static\_cast<int>(month)<10?"0":""), static\_cast<int>(month),

(day<10?"0":""), day,

hour<10?"0":"", hour,

min<10?"0":"", min,

sec<10?"0":"", sec );

return string(msg);

}

Date::Date(const Date &date) {

year = date.year;

month = date.month;

day = date.day;

hour = date.hour;

min = date.min;

sec = date.sec;

weekday = date.weekday;

time = (tm\*)malloc(sizeof(tm));

memcpy(time, date.time, sizeof(tm));

}

Date Date::addYears(int32\_t years) {

Date d = \*this;

d.year+=years;

if (d.year < -1 || d.year > 9999) throw std::runtime\_error("Year is out of range!");

reflect(d);

return d;

}

Date Date::addMonths(int32\_t months) {

Date d = \*this;

int32\_t m = static\_cast<typename std::underlying\_type<Month>::type>(d.month)+months;

d.year+=(m-1)/12;

if (d.year < -1 || d.year > 9999) throw std::runtime\_error("Year is out of range!");

d.month = static\_cast<Month>((m-1)%12+1);

reflect(d);

return d;

}

Date Date::addTime(int32\_t seconds) {

Date d = \*this;

seconds+=d.sec;

d.sec = seconds%60;

d.min+= seconds/1min;

d.hour+=d.min/60;

d.min%=60;

d.day+= d.hour/24;

d.hour%=24;

//int n = ;

int c = 0, n;

for(int i=0;i<2;i++){//hotfix

switch(d.month){ case Month::Jan: c = 31; break;

case Month::Feb: c = 28; if( (!(d.year%4) && !!(d.year%100) ) ||!(d.year%400))c = 29; break;

case Month::Mar: c = 31; break;

case Month::Apr: c = 30; break;

case Month::May: c = 31; break;

case Month::Jun: c = 30; break;

case Month::Jul: c = 31; break;

case Month::Aug: c = 31; break;

case Month::Sep: c = 30; break;

case Month::Oct: c = 31; break;

case Month::Nov: c = 30; break;

case Month::Dec: c = 31; break; }

n =static\_cast<typename std::underlying\_type<Month>::type>(d.month)+(d.day-1)/c;

n-=1;//hotfix

d.month = static\_cast<Month>((n-1)%12+1);}

d.year+=(n-1)/12;

if (d.year < -1 || d.year > 9999) throw std::runtime\_error("Year is out of range!");

// hotfix

int fix=d.day/c;

d.month = static\_cast<Month>(static\_cast<int>(d.month)+fix);

d.day%=c;

fix=static\_cast<int>(d.month)/12;

d.year+=fix;

d.month=static\_cast<Month>(static\_cast<int>(d.month)%12==0?12:static\_cast<int>(d.month)%12);

// \hotfix

reflect(d);

return d;

}

inline void Date::reflect(const Date& date) {

date.time->tm\_year = date.year-1900;

date.time->tm\_mon = static\_cast<typename std::underlying\_type<Month>::type>(date.month)-1;

date.time->tm\_mday = date.day-1;

date.time->tm\_wday = static\_cast<typename std::underlying\_type<Month>::type>(date.weekday)-1;

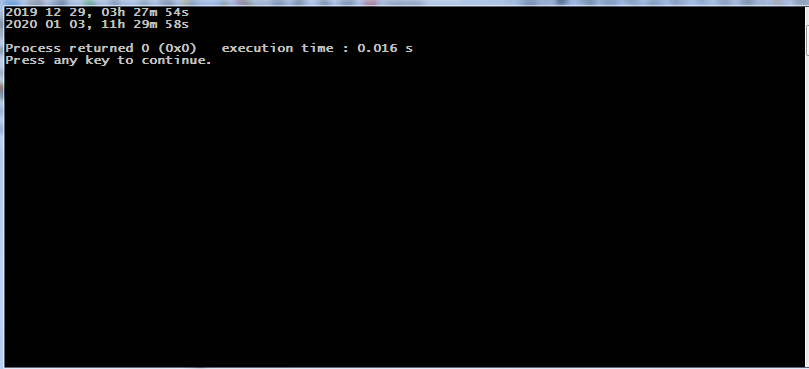
date.time->tm\_hour = date.hour;

date.time->tm\_min = date.min;

date.time->tm\_sec = date.sec;

}

**Результат работы программы:**

****