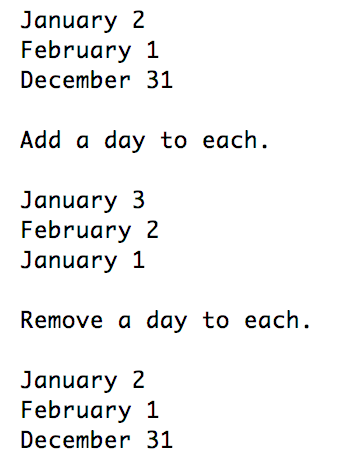
CSC 122 001 Computer Science II

Julius Ranoa

Chapter 11 Programming Challenge 3 Day of the Year Modification

Modify the *DayOfYear* class from the previous programming challenge and add a constructor that takes two parameters: a string representing a month and an integer in the range 0 through 31 representing the day of month. Also, add the prefix and postfix increment and decrement overloaded operators.

Screenshot of runtime.



Files included: (1) main.cpp, (2) DayOfYear.h, (3) DayOfYear.cpp

**main.cpp**

#include **<iostream>**#include **"DayOfYearMod.h"  
  
int** main() {  
  
 **int** test[ ] = {2, 32, 0}; *// Ignore last test case.* **const int** SIZE = **sizeof**(test) / **sizeof**(test[0]);  
  
 DayOfYearMod d[SIZE];  
  
 *// Assign test values.* **for** (**int** i = 0; i < SIZE - 1; i++) {  
 d[i].setDay(test[i]);  
 std::cout << d[i] << **" \n"**;  
 }  
 *// Assign last test case with new constructor* d[SIZE - 1] = DayOfYearMod(**"December"**, 31);  
 std::cout << d[SIZE - 1] << **"\n"**;  
  
 std::cout << **"\n"**;  
 std::cout << **"Add a day to each. \n"**;  
 std::cout << **"\n"**;  
  
 **for** (**int** i = 0; i < SIZE; i++) {  
 std::cout << ++d[i] << **"\n"**;  
 }  
  
 std::cout << **"\n"**;  
 std::cout << **"Remove a day to each. \n"**;  
 std::cout << **"\n"**;  
  
 **for** (**int** i = 0; i < SIZE; i++) {  
 std::cout << --d[i] << **"\n"**;  
 }  
  
 **return** 0;  
}

**DayOfYear.h**

#ifndef **CH11\_PR2\_DAY\_OF\_THE\_YEAR\_DAYOFYEAR\_H**#define **CH11\_PR2\_DAY\_OF\_THE\_YEAR\_DAYOFYEAR\_H**#include **<string>**#include **<iostream>  
  
class** DayOfYearMod {  
  
**public**:  
 **static const int** dayMax;  
 **static const int** numMonths;  
 **static int** daysPerMonth[ ]; *// One-based indexing. 1 = January* **static** std::string monthNames[ ];  
  
**private**:  
 **int** numDay; *// -nth day of the year  
  
 // Results* **int** numMonth;  
 std::string month;  
 **int** dayOfMonth;  
  
 **void** extractDetails();  
  
**public**:  
  
 DayOfYearMod();  
 DayOfYearMod(**int**);  
 DayOfYearMod(std::string, **int**);  
 **void** setDay(**const int**);  
  
 **bool** isInRange(**int**);  
 **void** addToDay(**int**);  
 **void** print();  
  
 *// Overloaded Operators* DayOfYearMod **operator**++(**int**);  
 DayOfYearMod& **operator**++();  
  
 DayOfYearMod **operator**--(**int**);  
 DayOfYearMod& **operator**--();  
  
 **friend** std::ostream& **operator**<<(std::ostream&, **const** DayOfYearMod&);  
};  
  
  
#endif *//CH11\_PR2\_DAY\_OF\_THE\_YEAR\_DAYOFYEAR\_H*

**DayOfYear.cpp**

*//  
// Created by TheLoneWoof on 2/6/18.  
//*#include **<algorithm>** *// For transform*#include **"DayOfYearMod.h"  
  
const int** DayOfYearMod::dayMax = 365;  
**const int** DayOfYearMod::numMonths = 12;  
  
*// One-based indexing. January = 1.  
// Assuming no leap years.***int** DayOfYearMod::daysPerMonth[DayOfYearMod::numMonths + 1] = {  
 0,  
 31, 28, 31, 30, 31, 30,  
 31, 31, 30, 31, 30, 31  
};  
  
std::string DayOfYearMod::monthNames[DayOfYearMod::numMonths + 1] = {  
 **""**,  
 **"January"**, **"February"**, **"March"**,  
 **"April"**, **"May"**, **"June"**,  
 **"July"**, **"August"**, **"September"**,  
 **"October"**, **"November"**, **"December"**};  
  
*// Constructors*DayOfYearMod::DayOfYearMod(**int** num) {  
 numDay = 0;  
 addToDay(num); *// This function allows for out-of-range values.*}  
  
*// Constructor delegated*DayOfYearMod::DayOfYearMod() : DayOfYearMod(1) { }  
  
DayOfYearMod::DayOfYearMod(std::string monthName, **int** day) {  
 **int** nMonth = 0;  
  
 *// Clean-up monthName for searching.* std::transform(monthName.begin(), monthName.end(), monthName.begin(), ::tolower);  
 monthName[0] = toupper(monthName[0]);  
  
 **for** (**int** i = 1; i <= numMonths; i++) {  
 **if** (monthName == monthNames[i]) {  
 nMonth = i;  
 **break**;  
 }  
 }  
  
 **if** (nMonth == 0) {  
 std::cout << **"Error: Month not found. "**;  
 exit(-404);  
 }  
  
 **if** (day <= 0 || day > daysPerMonth[nMonth]) {  
 std::cout << **"Error: Day out of range. "**;  
 exit(-403);  
 }  
  
 *// If month and day are valid.* numMonth = nMonth;  
 month = monthNames[nMonth];  
 dayOfMonth = day;  
  
 numDay = day;  
 **for** (**int** i = 1; i < numMonth; i++) {  
 numDay += daysPerMonth[i];  
 }  
}  
  
*// PRIVATE METHODS***void** DayOfYearMod::extractDetails() {  
 **int** temp = numDay;  
 **for** (**int** i = 1; i <= numMonths; i++) {  
 temp = temp - daysPerMonth[i];  
 **if** (temp <= 0) {  
 numMonth = i;  
 month = monthNames[i];  
 dayOfMonth = temp + daysPerMonth[i];  
 **break**;  
 }  
 }  
}  
  
**void** DayOfYearMod::addToDay(**int** plus) {  
 **this**->numDay = (**this**->numDay + plus) % DayOfYearMod::dayMax;  
 *// If r == 0, then the day is the max day.* **if** (**this**->numDay == 0) **this**->numDay = DayOfYearMod::dayMax;  
 **this**->extractDetails();  
}  
  
*// PUBLIC METHODS***bool** DayOfYearMod::isInRange(**int** val) {  
 **if** (val > 0 && val <= dayMax) {  
 **return true**;  
 } **else return false**;  
}  
  
**void** DayOfYearMod::setDay(**const int** val) {  
 **if** (!isInRange(val)) {  
 std::cout << **"Day not in range."**;  
 exit(-1);  
 }  
 numDay = val;  
 extractDetails();  
}  
  
**void** DayOfYearMod::print() {  
 std::cout << month << **" "** << dayOfMonth;  
}  
  
*// OVERLOADED OPERATORS  
  
// Prefix Increment*DayOfYearMod& DayOfYearMod::**operator**++() {  
 **this**->addToDay(1);  
 **return** \***this**;  
}  
  
*// Postfix Increment  
// This postfix operator functions the same as the default  
// postfix operator, meaning that a copy of the object is  
// returned and the current object is incremented.*DayOfYearMod DayOfYearMod::**operator**++(**int**) {  
 DayOfYearMod copy(\***this**);  
 **this**->addToDay(1);  
 **return** copy;  
}  
  
*// Prefix Decrement*DayOfYearMod& DayOfYearMod::**operator**--() {  
 **this**->addToDay(-1);  
 **return** \***this**;  
}  
  
*// Postfix Decrement  
// See note on postfix increment.*DayOfYearMod DayOfYearMod::**operator**--(**int**) {  
 DayOfYearMod copy(\***this**);  
 **this**->addToDay(-1);  
 **return** copy;  
}  
  
*// Output stream operator*std::ostream& **operator**<<(std::ostream& out, **const** DayOfYearMod& d) {  
 out << d.month << **" "** << d.dayOfMonth;  
 **return** out;  
}