**Exercise 10**

**Develop the following:**

**1.** Create a class called *Date* that represents any day, month and year of a particular date.

**2.** Make sure that the *day* and *year* member variables are integer numbers and *month* is an enumerated data type, like so:

enum class Month

{

January = 1,

February,

March,

April,

May,

June,

July,

August,

September,

October,

November,

December

};

**3.** Add a constructor with parameters representing the *day*, *month* and *year* and add some default arguments.

**4.** Add a setter function to set the *day*, *month* and *year* to a specific date. Make every effort to ascertain the date is formatted correctly. For example, days larger than **31** or negative years should not be accepted.

**5.** Add a member function that checks if the year of a given date is a *leap year* or not. Use the following as a reference for calculating leap years:

<https://www.mathsisfun.com/leap-years.html>

**6.** Overload the **++** and **--** operator, both in *prefix* and *postfix* form. Both overloads should increase/decrease the date by one single day. Remember that each month has a different number of days and cannot be less than 1. When we reach the end of the year, and we increment, we must increment the year as well. Likewise, when we are at the start of the year, and we decrement, we also go back one year. Moreover, February has **29** days if it’s a leap year, so you have to factor that in as well. The operator overloads should allow us to write the following code in *main.cpp*:

Date specialDay(4, Date::Month::July, 1982);

++specialDay;

specialDay--;

**7.** Overload the **+** and **-** operator such that we can advance or retract a date by a specific scalar amount. Take into account the factors mentioned in question **6**. We should be able to write the following code in *main.cpp*:

Date today(26, Date::Month::July, 2022);

Date dayAfterTomorrow = today + 2;

Date lastWeek = today - 7;

**8.** Overload the equality operators (**==** and **!=**) and make sure they check if two dates are exactly the same or not the same. You may choose to compare the day, month and year or only compare days or only years. For example, the following code could be allowed:

Date aprilFoolsDay(1, Date::Month::April, 2019);

if (aprilFoolsDay == Date::Month::April)

{

//The date is in the month of April

}

**OR**

Date myBirthday(30, Date::Month::March, 1980);

Date yourBirthday(1, Date::Month::May, 1995);

if (myBirthday == yourBirthday)

{

//The birthdays are the same

}

**9.** Overload the relational operators (**>**, **<** ,**>=**, **<=**) and try to compare one *Date* object with another and check if the day, month and year are less than, or greater than, the passed *Date* object.

**10.** Add a member function that converts the given date into a *std::string* type text, so that the date is formatted and readable. The function should pay special attention to the day such that any day with a **1** has a **st** at the end. Similarly, **2** would have a **nd** and **3** would have **rd**. Below are some examples:

**1st of April 1998**

**23rd of May 2007**

**22nd of December 1800**