```
#pragma warning(disable : 4996)
#pragma once

#include<iostream>
#include<string>
#include "clsString.h"

using namespace std;

class clsDate
{

private:

    short _Day = 1;
    short _Month = 1;
    short _Year = 1900;

public:

    clsDate()
    {
        time_t t = time(0);
        tm* now = localtime(&t);
        _Day = now->tm_mday;
        _Month = now->tm_mon + 1;
        _Year = now->tm_year + 1900;
    }

    clsDate(string sDate)
    {

        vector <string> vDate;
        vDate = clsString::Split(sDate, "/");

        _Day = stoi(vDate[0]);
        _Month = stoi(vDate[1]);
        _Year = stoi(vDate[2]);

    }

    clsDate(short Day, short Month, short Year)
    {

        _Day = Day;
        _Month = Month;
        _Year = Year;

    }

    clsDate(short DateOrderInYear, short Year)
    {
        //This will construct a date by date order in year
```

```
        clsDate Date1 = GetDateFromDayOrderInYear(DateOrderInYear, Year);
        _Day = Date1.Day;
        _Month = Date1.Month;
        _Year = Date1.Year;
    }

    void SetDay(short Day) {
        _Day = Day;
    }

    short GetDay() {
        return _Day;
    }
    __declspec(property(get = GetDay, put = SetDay)) short Day;

    void SetMonth(short Month) {
        _Month = Month;
    }

    short GetMonth() {
        return _Month;
    }
    __declspec(property(get = GetMonth, put = SetMonth)) short Month;


    void SetYear(short Year) {
        _Year = Year;
    }

    short GetYear() {
        return _Year;
    }
    __declspec(property(get = GetYear, put = SetYear)) short Year;

    void Print()
    {
        cout << DateToString() << endl;
    }

    static clsDate GetSystemDate()
    {
        //system date
        time_t t = time(0);
        tm* now = localtime(&t);

        short Day, Month, Year;

        Year = now->tm_year + 1900;
        Month = now->tm_mon + 1;
        Day = now->tm_mday;

        return clsDate(Day, Month, Year);
```

```
    }

    static  bool IsValidDate(clsDate Date)
    {
        if (Date.Day < 1 || Date.Day>31)
            return false;

        if (Date.Month < 1 || Date.Month>12)
            return false;

        if (Date.Month == 2)
        {
            if (isLeapYear(Date.Year))
            {
                if (Date.Day > 29)
                    return false;
            }
            else
            {
                if (Date.Day > 28)
                    return false;
            }
        }

        short DaysInMonth = NumberOfDaysInAMonth(Date.Month, Date.Year);

        if (Date.Day > DaysInMonth)
            return false;

        return true;

    }

    bool IsValid()
    {
        return IsValidDate(*this);
    }

    static string DateToString(clsDate Date)
    {
        return  to_string(Date.Day) + "/" + to_string(Date.Month) + "/" + to_string(Date.Year);
    }

    string DateToString()
    {
        return  DateToString(*this);
    }

    static bool isLeapYear(short Year)
    {
        // if year is divisible by 4 AND not divisible by 100
```

```
        // OR if year is divisible by 400
        // then it is a leap year
          return (Year % 4 == 0 && Year % 100 != 0) || (Year % 400 ==
            0);
      }

      bool isLeapYear()
      {
          return isLeapYear(_Year);
      }

      static short NumberOfDaysInAYear(short Year)
      {
          return  isLeapYear(Year) ? 365 : 364;
      }

      short NumberOfDaysInAYear()
      {
          return  NumberOfDaysInAYear(_Year);
      }

      static short NumberOfHoursInAYear(short Year)
      {
          return  NumberOfDaysInAYear(Year) * 24;
      }

      short NumberOfHoursInAYear()
      {
          return  NumberOfHoursInAYear(_Year);
      }

      static int NumberOfMinutesInAYear(short Year)
      {
          return  NumberOfHoursInAYear(Year) * 60;
      }

      int NumberOfMinutesInAYear()
      {
          return  NumberOfMinutesInAYear(_Year);
      }

      static int NumberOfSecondsInAYear(short Year)
      {
          return  NumberOfMinutesInAYear(Year) * 60;
      }

      int NumberOfSecondsInAYear()
      {
          return  NumberOfSecondsInAYear();
      }

      static short NumberOfDaysInAMonth(short Month, short Year)
      {
```

```
        if (Month < 1 || Month>12)
            return  0;

        int days[12] = { 31,28,31,30,31,30,31,31,30,31,30,31 };
        return (Month == 2) ? (isLeapYear(Year) ? 29 : 28) : days
          [Month - 1];

    }

    short NumberOfDaysInAMonth()
    {
        return NumberOfDaysInAMonth(_Month, _Year);
    }

    static short NumberOfHoursInAMonth(short Month, short Year)
    {
        return  NumberOfDaysInAMonth(Month, Year) * 24;
    }

    short NumberOfHoursInAMonth()
    {
        return  NumberOfDaysInAMonth(_Month, _Year) * 24;
    }

    static int NumberOfMinutesInAMonth(short Month, short Year)
    {
        return  NumberOfHoursInAMonth(Month, Year) * 60;
    }

    int NumberOfMinutesInAMonth()
    {
        return  NumberOfHoursInAMonth(_Month, _Year) * 60;
    }

    static int NumberOfSecondsInAMonth(short Month, short Year)
    {
        return  NumberOfMinutesInAMonth(Month, Year) * 60;
    }

    int NumberOfSecondsInAMonth()
    {
        return  NumberOfMinutesInAMonth(_Month, _Year) * 60;
    }

    static short DayOfWeekOrder(short Day, short Month, short Year)
    {
        short a, y, m;
        a = (14 - Month) / 12;
        y = Year - a;
        m = Month + (12 * a) - 2;
        // Gregorian:
        //0:sun, 1:Mon, 2:Tue...etc
```

```
        return (Day + y + (y / 4) - (y / 100) + (y / 400) + ((31 * m) / 12)) % 7;
    }

    short DayOfWeekOrder()
    {
        return DayOfWeekOrder(_Day, _Month, _Year);
    }

    static string DayShortName(short DayOfWeekOrder)
    {
        string arrDayNames[] = { "Sun","Mon","Tue","Wed","Thu","Fri","Sat" };

        return arrDayNames[DayOfWeekOrder];
    }

    static string DayShortName(short Day, short Month, short Year)
    {
        string arrDayNames[] = { "Sun","Mon","Tue","Wed","Thu","Fri","Sat" };

        return arrDayNames[DayOfWeekOrder(Day, Month, Year)];
    }

    string DayShortName()
    {
        string arrDayNames[] = { "Sun","Mon","Tue","Wed","Thu","Fri","Sat" };

        return arrDayNames[DayOfWeekOrder(_Day, _Month, _Year)];
    }

    static string MonthShortName(short MonthNumber)
    {
        string Months[12] = { "Jan", "Feb", "Mar",
                              "Apr", "May", "Jun",
                              "Jul", "Aug", "Sep",
                              "Oct", "Nov", "Dec"
        };

        return (Months[MonthNumber - 1]);
    }

    string MonthShortName()
    {
        return MonthShortName(_Month);
    }

    static void PrintMonthCalendar(short Month, short Year)
    {
        int NumberOfDays;

```

```
        // Index of the day from 0 to 6
        int current = DayOfWeekOrder(1, Month, Year);

        NumberOfDays = NumberOfDaysInAMonth(Month, Year);

        // Print the current month name
        printf("\n  _______________%s_______________\n\n",
            MonthShortName(Month).c_str());

        // Print the columns
        printf("  Sun  Mon  Tue  Wed  Thu  Fri  Sat\n");

        // Print appropriate spaces
        int i;
        for (i = 0; i < current; i++)
            printf("     ");

        for (int j = 1; j <= NumberOfDays; j++)
        {
            printf("%5d", j);


            if (++i == 7)
            {
                i = 0;
                printf("\n");
            }
        }

        printf("\n  _________________________________\n");
    }

    void PrintMonthCalendar()
    {
        PrintMonthCalendar(_Month, _Year);
    }

    static void PrintYearCalendar(int Year)
    {
        printf("\n  _________________________________\n\n");
        printf("           Calendar - %d\n", Year);
        printf("  _________________________________\n");


        for (int i = 1; i <= 12; i++)
        {
            PrintMonthCalendar(i, Year);
        }

        return;
    }

    void PrintYearCalendar()
```

```
    {
        printf("\n  _________________________________\n\n");
        printf("           Calendar - %d\n", _Year);
        printf("  _________________________________\n");


        for (int i = 1; i <= 12; i++)
        {
            PrintMonthCalendar(i, _Year);
        }

        return;
    }

    static short DaysFromTheBeginingOfTheYear(short Day, short Month, short Year)
    {
        short TotalDays = 0;

        for (int i = 1; i <= Month - 1; i++)
        {
            TotalDays += NumberOfDaysInAMonth(i, Year);
        }

        TotalDays += Day;

        return TotalDays;
    }

    short DaysFromTheBeginingOfTheYear()
    {


        short TotalDays = 0;

        for (int i = 1; i <= _Month - 1; i++)
        {
            TotalDays += NumberOfDaysInAMonth(i, _Year);
        }

        TotalDays += _Day;

        return TotalDays;
    }

    static clsDate GetDateFromDayOrderInYear(short DateOrderInYear, short Year)
    {
        clsDate Date;
        short RemainingDays = DateOrderInYear;
        short MonthDays = 0;

        Date.Year = Year;
```

```
        Date.Month = 1;

        while (true)
        {
            MonthDays = NumberOfDaysInAMonth(Date.Month, Year);

            if (RemainingDays > MonthDays)
            {
                RemainingDays -= MonthDays;
                Date.Month++;
            }
            else
            {
                Date.Day = RemainingDays;
                break;
            }
        }

        return Date;
    }

    void AddDays(short Days)
    {
        short RemainingDays = Days + DaysFromTheBeginingOfTheYear
          (_Day, _Month, _Year);
        short MonthDays = 0;

        _Month = 1;

        while (true)
        {
            MonthDays = NumberOfDaysInAMonth(_Month, _Year);

            if (RemainingDays > MonthDays)
            {
                RemainingDays -= MonthDays;
                _Month++;

                if (_Month > 12)
                {
                    _Month = 1;
                    _Year++;

                }
            }
            else
            {
                _Day = RemainingDays;
                break;
            }
        }
    }

```

```
    static bool IsDate1BeforeDate2(clsDate Date1, clsDate Date2)
    {
        return  (Date1.Year < Date2.Year) ? true : ((Date1.Year == Date2.Year) ? (Date1.Month < Date2.Month ? true : (Date1.Month == Date2.Month ? Date1.Day < Date2.Day : false)) : false);
    }

    bool IsDateBeforeDate2(clsDate Date2)
    {
        //note: *this sends the current object :-)
        return  IsDate1BeforeDate2(*this, Date2);  //the same of IsDate1BeforeDate2(d1, d2)
    }

    static bool IsDate1EqualDate2(clsDate Date1, clsDate Date2)
    {
        return  (Date1.Year == Date2.Year) ? ((Date1.Month == Date2.Month) ? ((Date1.Day == Date2.Day) ? true : false) : false) : false;
    }

    bool IsDateEqualDate2(clsDate Date2)
    {
        return  IsDate1EqualDate2(*this, Date2);
    }

    static bool IsLastDayInMonth(clsDate Date)
    {
        return (Date.Day == NumberOfDaysInAMonth(Date.Month, Date.Year));
    }

    bool IsLastDayInMonth()
    {
        return IsLastDayInMonth(*this);
    }

    static bool IsLastMonthInYear(short Month)
    {
        return (Month == 12);
    }

    static clsDate AddOneDay(clsDate Date)
    {
        if (IsLastDayInMonth(Date))
        {
            if (IsLastMonthInYear(Date.Month))
            {
                Date.Month = 1;
                Date.Day = 1;
                Date.Year++;
            }
```

```
            else
            {
                Date.Day = 1;
                Date.Month++;
            }
        }
        else
        {
            Date.Day++;
        }

        return Date;
    }

    void AddOneDay()
    {
        *this = AddOneDay(*this);
    }

    static void  SwapDates(clsDate& Date1, clsDate& Date2)
    {
        clsDate TempDate;
        TempDate = Date1;
        Date1 = Date2;
        Date2 = TempDate;
    }

    static int GetDifferenceInDays(clsDate Date1, clsDate Date2, bool IncludeEndDay = false)
    {
        //this will take care of negative diff
        int Days = 0;
        short SawpFlagValue = 1;

        if (!IsDate1BeforeDate2(Date1, Date2))
        {
            //Swap Dates
            SwapDates(Date1, Date2);
            SawpFlagValue = -1;

        }

        while (IsDate1BeforeDate2(Date1, Date2))
        {
            Days++;
            Date1 = AddOneDay(Date1);
        }

        return IncludeEndDay ? ++Days * SawpFlagValue : Days * SawpFlagValue;
    }

    int GetDifferenceInDays(clsDate Date2, bool IncludeEndDay = false)
```

```
    {
        return GetDifferenceInDays(*this, Date2, IncludeEndDay);
    }

    static short CalculateMyAgeInDays(clsDate DateOfBirth)
    {
        return GetDifferenceInDays(DateOfBirth, clsDate::GetSystemDate ↵
          (), true);
    }
    //above no need to have nonstatic function for the object because  ↵
      it does not depend on any data from it.

    static clsDate IncreaseDateByOneWeek(clsDate& Date)
    {
        for (int i = 1; i <= 7; i++)
        {
            Date = AddOneDay(Date);
        }

        return Date;
    }

    void IncreaseDateByOneWeek()
    {
        IncreaseDateByOneWeek(*this);
    }

    clsDate IncreaseDateByXWeeks(short Weeks, clsDate& Date)
    {
        for (short i = 1; i <= Weeks; i++)
        {
            Date = IncreaseDateByOneWeek(Date);
        }
        return Date;
    }

    void IncreaseDateByXWeeks(short Weeks)
    {
        IncreaseDateByXWeeks(Weeks, *this);
    }

    clsDate IncreaseDateByOneMonth(clsDate& Date)
    {
        if (Date.Month == 12)
        {
            Date.Month = 1;
            Date.Year++;
        }
        else
        {
            Date.Month++;
        }

```

```
    //last check day in date should not exceed max days in the current month
        // example if date is 31/1/2022 increasing one month should not be 31/2/2022, it should
        // be 28/2/2022
        short NumberOfDaysInCurrentMonth = NumberOfDaysInAMonth(Date.Month, Date.Year);
        if (Date.Day > NumberOfDaysInCurrentMonth)
        {
            Date.Day = NumberOfDaysInCurrentMonth;
        }

        return Date;
    }

    void IncreaseDateByOneMonth()
    {
        IncreaseDateByOneMonth(*this);
    }

    clsDate IncreaseDateByXDays(short Days, clsDate& Date)
    {

        for (short i = 1; i <= Days; i++)
        {
            Date = AddOneDay(Date);
        }
        return Date;
    }

    void IncreaseDateByXDays(short Days)
    {

        IncreaseDateByXDays(Days, *this);
    }

    clsDate IncreaseDateByXMonths(short Months, clsDate& Date)
    {
        for (short i = 1; i <= Months; i++)
        {
            Date = IncreaseDateByOneMonth(Date);
        }
        return Date;
    }

    void IncreaseDateByXMonths(short Months)
    {
        IncreaseDateByXMonths(Months, *this);
    }

    static clsDate IncreaseDateByOneYear(clsDate& Date)
    {
        Date.Year++;
```

```
        return Date;
    }

    void IncreaseDateByOneYear()
    {
        IncreaseDateByOneYear(*this);
    }

    clsDate IncreaseDateByXYears(short Years, clsDate& Date)
    {
        Date.Year += Years;
        return Date;

    }

    void IncreaseDateByXYears(short Years)
    {
        IncreaseDateByXYears(Years);
    }

    clsDate IncreaseDateByOneDecade(clsDate& Date)
    {
        //Period of 10 years
        Date.Year += 10;
        return Date;
    }

    void IncreaseDateByOneDecade()
    {
        IncreaseDateByOneDecade(*this);
    }

    clsDate IncreaseDateByXDecades(short Decade, clsDate& Date)
    {
        Date.Year += Decade * 10;
        return Date;
    }

    void IncreaseDateByXDecades(short Decade)
    {
        IncreaseDateByXDecades(Decade, *this);
    }

    clsDate IncreaseDateByOneCentury(clsDate& Date)
    {
        //Period of 100 years
        Date.Year += 100;
        return Date;
    }

    void IncreaseDateByOneCentury()
    {
        IncreaseDateByOneCentury(*this);
```

```
    }

    clsDate IncreaseDateByOneMillennium(clsDate& Date)
    {
        //Period of 1000 years
        Date.Year += 1000;
        return Date;
    }

    clsDate IncreaseDateByOneMillennium()
    {
        IncreaseDateByOneMillennium(*this);
    }

    static clsDate DecreaseDateByOneDay(clsDate Date)
    {
        if (Date.Day == 1)
        {
            if (Date.Month == 1)
            {
                Date.Month = 12;
                Date.Day = 31;
                Date.Year--;
            }
            else
            {

                Date.Month--;
                Date.Day = NumberOfDaysInAMonth(Date.Month, Date.Year);
            }
        }
        else
        {
            Date.Day--;
        }

        return Date;
    }

    void DecreaseDateByOneDay()
    {
        DecreaseDateByOneDay(*this);
    }

    static clsDate DecreaseDateByOneWeek(clsDate& Date)
    {
        for (int i = 1; i <= 7; i++)
        {
            Date = DecreaseDateByOneDay(Date);
        }

        return Date;
```

```
    }

    void DecreaseDateByOneWeek()
    {
        DecreaseDateByOneWeek(*this);
    }

    static clsDate DecreaseDateByXWeeks(short Weeks, clsDate& Date)
    {
        for (short i = 1; i <= Weeks; i++)
        {
            Date = DecreaseDateByOneWeek(Date);
        }
        return Date;
    }

    void DecreaseDateByXWeeks(short Weeks)
    {
        DecreaseDateByXWeeks(Weeks, *this);
    }

    static clsDate DecreaseDateByOneMonth(clsDate& Date)
    {

        if (Date.Month == 1)
        {
            Date.Month = 12;
            Date.Year--;
        }
        else
            Date.Month--;


        //last check day in date should not exceed max days in the current month
       // example if date is 31/3/2022 decreasing one month should not be 31/2/2022, it should
       // be 28/2/2022
        short NumberOfDaysInCurrentMonth = NumberOfDaysInAMonth(Date.Month, Date.Year);
        if (Date.Day > NumberOfDaysInCurrentMonth)
        {
            Date.Day = NumberOfDaysInCurrentMonth;
        }


        return Date;
    }

    void DecreaseDateByOneMonth()
    {
        DecreaseDateByOneMonth(*this);
    }
```

```
    static clsDate DecreaseDateByXDays(short Days, clsDate& Date)
    {

        for (short i = 1; i <= Days; i++)
        {
            Date = DecreaseDateByOneDay(Date);
        }
        return Date;
    }

    void DecreaseDateByXDays(short Days)
    {
        DecreaseDateByXDays(Days, *this);
    }

    static clsDate DecreaseDateByXMonths(short Months, clsDate& Date)
    {

        for (short i = 1; i <= Months; i++)
        {
            Date = DecreaseDateByOneMonth(Date);
        }
        return Date;
    }

    void DecreaseDateByXMonths(short Months)
    {
        DecreaseDateByXMonths(Months, *this);
    }

    static clsDate DecreaseDateByOneYear(clsDate& Date)
    {

        Date.Year--;
        return Date;
    }

    void DecreaseDateByOneYear()
    {
        DecreaseDateByOneYear(*this);
    }

    static clsDate DecreaseDateByXYears(short Years, clsDate& Date)
    {

        Date.Year -= Years;
        return Date;
    }

    void DecreaseDateByXYears(short Years)
    {
        DecreaseDateByXYears(Years, *this);
```

```
    }

    static clsDate DecreaseDateByOneDecade(clsDate& Date)
    {
        //Period of 10 years
        Date.Year -= 10;
        return Date;
    }

    void DecreaseDateByOneDecade()
    {
        DecreaseDateByOneDecade(*this);
    }

    static clsDate DecreaseDateByXDecades(short Decades, clsDate&
      Date)
    {
        Date.Year -= Decades * 10;
        return Date;
    }

    void DecreaseDateByXDecades(short Decades)
    {
        DecreaseDateByXDecades(Decades, *this);
    }

    static clsDate DecreaseDateByOneCentury(clsDate& Date)
    {
        //Period of 100 years
        Date.Year -= 100;
        return Date;
    }

    void DecreaseDateByOneCentury()
    {
        DecreaseDateByOneCentury(*this);
    }

    static clsDate DecreaseDateByOneMillennium(clsDate& Date)
    {
        //Period of 1000 years
        Date.Year -= 1000;
        return Date;
    }

    void DecreaseDateByOneMillennium()
    {
        DecreaseDateByOneMillennium(*this);
    }

    static short IsEndOfWeek(clsDate Date)
    {
        return  DayOfWeekOrder(Date.Day, Date.Month, Date.Year) == 6;
```

```
    }

    short IsEndOfWeek()
    {
        return IsEndOfWeek(*this);
    }

    static bool IsWeekEnd(clsDate Date)
    {
        //Weekends are Fri and Sat
        short DayIndex = DayOfWeekOrder(Date.Day, Date.Month, Date.Year);
        return  (DayIndex == 5 || DayIndex == 6);
    }

    bool IsWeekEnd()
    {
        return  IsWeekEnd(*this);
    }

    static bool IsBusinessDay(clsDate Date)
    {
        //Weekends are Sun,Mon,Tue,Wed and Thur

       /*
        short DayIndex = DayOfWeekOrder(Date.Day, Date.Month, Date.Year);
        return  (DayIndex >= 5 && DayIndex <= 4);
       */

       //shorter method is to invert the IsWeekEnd: this will save updating code.
        return !IsWeekEnd(Date);
    }

    bool IsBusinessDay()
    {
        return  IsBusinessDay(*this);
    }

    static short DaysUntilTheEndOfWeek(clsDate Date)
    {
        return 6 - DayOfWeekOrder(Date.Day, Date.Month, Date.Year);
    }

    short DaysUntilTheEndOfWeek()
    {
        return  DaysUntilTheEndOfWeek(*this);
    }

    static short DaysUntilTheEndOfMonth(clsDate Date1)
    {
        clsDate EndOfMontDate;
```

```
        EndOfMontDate.Day = NumberOfDaysInAMonth(Date1.Month, Date1.Year);
        EndOfMontDate.Month = Date1.Month;
        EndOfMontDate.Year = Date1.Year;

        return GetDifferenceInDays(Date1, EndOfMontDate, true);
    }

    short DaysUntilTheEndOfMonth()
    {
        return DaysUntilTheEndOfMonth(*this);
    }

    static short DaysUntilTheEndOfYear(clsDate Date1)
    {
        clsDate EndOfYearDate;
        EndOfYearDate.Day = 31;
        EndOfYearDate.Month = 12;
        EndOfYearDate.Year = Date1.Year;

        return GetDifferenceInDays(Date1, EndOfYearDate, true);
    }

    short DaysUntilTheEndOfYear()
    {
        return  DaysUntilTheEndOfYear(*this);
    }

    //i added this method to calculate business days between 2 days
    static short CalculateBusinessDays(clsDate DateFrom, clsDate DateTo)
    {
        short Days = 0;
        while (IsDate1BeforeDate2(DateFrom, DateTo))
        {
            if (IsBusinessDay(DateFrom))
                Days++;

            DateFrom = AddOneDay(DateFrom);
        }

        return Days;
    }

    static short CalculateVacationDays(clsDate DateFrom, clsDate DateTo)
    {
        /*short Days = 0;
        while (IsDate1BeforeDate2(DateFrom, DateTo))
        {
            if (IsBusinessDay(DateFrom))
                Days++;

```

```
            DateFrom = AddOneDay(DateFrom);
        }*/

        return CalculateBusinessDays(DateFrom, DateTo);
    }
    //above method is eough , no need to have method for the object

    static clsDate CalculateVacationReturnDate(clsDate DateFrom, short VacationDays)
    {

        short WeekEndCounter = 0;

        for (short i = 1; i <= VacationDays; i++)
        {

            if (IsWeekEnd(DateFrom))
                WeekEndCounter++;

            DateFrom = AddOneDay(DateFrom);
        }
        //to add weekends
        for (short i = 1; i <= WeekEndCounter; i++)
            DateFrom = AddOneDay(DateFrom);

        return DateFrom;
    }

    static bool IsDate1AfterDate2(clsDate Date1, clsDate Date2)
    {
        return (!IsDate1BeforeDate2(Date1, Date2) && !IsDate1EqualDate2(Date1, Date2));

    }

    bool IsDateAfterDate2(clsDate Date2)
    {
        return IsDate1AfterDate2(*this, Date2);
    }

    enum enDateCompare { Before = -1, Equal = 0, After = 1 };

    static enDateCompare CompareDates(clsDate Date1, clsDate Date2)
    {
        if (IsDate1BeforeDate2(Date1, Date2))
            return enDateCompare::Before;

        if (IsDate1EqualDate2(Date1, Date2))
            return enDateCompare::Equal;

        /* if (IsDate1AfterDate2(Date1,Date2))
             return enDateCompare::After;*/

```

```
            //this is faster
        return enDateCompare::After;
    }

    enDateCompare CompareDates(clsDate Date2)
    {
        return CompareDates(*this, Date2);
    }
};

/*
Explain for time constructure:
قبل ما تبلشي قراءة, هاد الشرح موجود بفيديو عند الفيديوهات باللابتوب

clsDate() {
    time_t t = time(0);
    tm* now = localtime(&t);
    _Day = now->tm_mday;
    _Month = now->tm_mon + 1;
    _Year = now->tm_year + 1900;
}

◇ 1. What is time_t t?
time_t is just a data type (like int or long long) made for storing time values.
Different systems may define it as long, long long, etc.
Here, t is a variable 🗄 that will hold the number of seconds

◇ 2. What does time(0) mean?
time() is a function that gives you the current system time.
It counts the seconds from a special starting point called the epoch.
👉 The -epoch- is January 1st, 1970, at 00:00:00 UTC.
If you call time(0) today (Sep 6, 2025)(هاد تاريخ اليوم), it might return something like 1757131000.
That number means: 1,757,131,000 seconds have passed since Jan 1, 1970.

time(0) حطينا صفر داخل
لأنه بدي إياه يرجّعلي عدد الثواني اللي مرّت من
1/Jan/ 1970
وممكن كمان تحطي
NULL
بدل الصفر لأنه في هذا السياق نفس المعنى. وإذا بدك، فيكي
& تعملي متغير وتبعثي عنوانه باستخدام
عشان الدالة تخزّن النتيجة فيه. بس أنا هون بدي
القيمة مباشرة، لهيك مرّرت صفر


◇ ليش بدأوا العد من 1/1/1970 وليس من بداية العام 1 ميلادي أو بداية أي سنة ثانية؟

1) ⏱ :الكمبيوتر يحتاج نقطة بداية ثابتة
-كل نظام تشغيل يحتاج يوم محدد يكون صفر
```

-UNIX اختارت 1/1/1970 لأنها كانت مناسبة وقت تطويره

2) : عشان الأرقام تكون صغيرة 💾
-إذا بدأنا العد من السنة 1، الأرقام بتكون ضخمة جدًا (>60 مليار ثانية!)ت
-من 1970
الأرقام أصغر وتناسب عدد البتات
32-bits
اللي كانت شائعة وقتها

3) : عشان يكون معيار موحّد🌍
حددت 1/1/1970 كمعيار UNIX.
باقي الأنظمة اتبعت نفس الشيء عشان التوافق (Linux, macOS, C++...)

◇ مثال بسيط
إذا بدأنا العد من السنة 1: يعني 1/1/1
وكان تاريخ اليوم = 6/9/2025
عدد الثواني = أكثر من 63 مليار ثانية! 😱
لكن إذا بدأنا العد من 1/1/1970:
وكان تاريخ اليوم = 6/9/2025
عدد الثواني ≈ 1.7 مليار فقط ☑ أسهل للكمبيوتر يتعامل معها.

◇ : الخلاصة

اخترنا 1970 لأنها كانت بداية UNIX وعملي للمبرمجين وقتها.

بدء العد من سنة 1 يجعل الأرقام ضخمة جدًا ويصعب التعامل معها.

يعتمد على هذه البداية لأنها معيار عالمي الآن C++ / UNIX نظام.

كيف الكمبيوتر بحسب التاريخ؟
◇ ليش من 1970؟

احنا قررنا نبدأ العدّ من 1/1/1970 ونسميه الصفر (Epoch)
يعني:
1/1/1970 = 0 ثانية.
بعد ثانية وحدة = 1.
بعد دقيقة وحدة = 60.
بعد ساعة = 3600.
بعد يوم = 86400

◇ كيف الكمبيوتر يعرف يعدّ؟
الكمبيوتر عنده ساعة داخلية (زي ساعة الحائط ⏰ بس دقيقة جدًا) 1)
🔋 الساعة بتضل شغّالة حتى لو الجهاز مطفي (عشان فيها بطارية صغيرة) 2)
بيوخد الوقت الحالي من الساعة (Windows, Linux, …) نظام التشغيل 3).
بعدين النظام بيحوّل الوقت الحالي إلى:
👉 كم يوم مرّ من "1/1/1970 × 86400 + الساعات + الدقايق + الثواني".

◇ أمثلة بسيطة

مثال 1: 2 يناير 1970 الساعة 00:00:00 🗓️
مرّ يوم واحد من 1/1/1970.
عدد الثواني = 1 × 86400 = 86400 ثانية.

مثال 2: 1 يناير 1970 الساعة 01:00:00 🗓️
مرّت ساعة وحدة.
عدد الثواني = 1 × 3600 = 3600 ثانية.
.

مثال 3: 6 سبتمبر 2025 الساعة 00:00:00 🗓️
طبعًا الكمبيوتر يحسب السنين الكاملة + الشهور + الأيام + الساعات + (الدقايق + الثواني).
لو نحسبها، بيكون الناتج حوالي: 1,757,132,400 ثانية من 1/1/1970.

الخلاصة : ✨
البداية (0) = 1/1/1970
الكمبيوتر يعدّ الأيام والساعات والدقايق من ذاك اليوم.
وبكل مرة بدنا "التاريخ"، النظام يعمل العملية بالعكس (يرجع يحوّل الثواني → تاريخ).

◇ 1. tm* now = localtime(&t);
t(UNIX time) فيه عدد الثواني منذ 1/1/1970
يحوّل هذا الرقم الكبير إلى تاريخ وساعة مفهومة للبشر localtime(&t)
tm) يحتوي: اسمه (struct) النتيجة هي هيكل بيانات:
tm_mday → (1–31) اليوم
tm_mon → (0 = يناير) ⚠ (0–11) الشهر
tm_year → 1900 منذ عدد السنوات
بالإضافة لساعات، دقائق، ثواني
tm* now = مؤشر (pointer) يشير لهذا الهيكل

◇ 2. _Day = now->tm_mday;
tm_mday (1–31) يعطيك رقم اليوم من الشهر
نخزّنه في _Day داخل الكلاس.
مثال:
إذا اليوم 6 سبتمبر → _Day = 6

◇ 3. _Month = now->tm_mon + 1;
tm_mon يبدأ من 0 → 0 = يناير، 1 = فبراير…
نضيف +1 علشان يكون طبيعي (1–12)
مثال:
tm_mon = 8 سبتمبر → _Month = 8 + 1 = 9 في

◇ 4. _Year = now->tm_year + 1900;
tm_year = 1900 منذ عدد السنوات
نضيف 1900 للحصول على السنة الفعلية
مثال:
إذا tm_year = 125 → _Year = 125 + 1900 = 2025

◇ الخلاصة البسيطة
localtime يحوّل الثواني منذ 1970 → تاريخ قابل للقراءة
tm نقرأ اليوم، الشهر، والسنة من الهيكل
نضيف +1 للشهر و +1900 للسنة لتصبح طبيعية

```
الآن _Day, _Month, _Year جاهزين للاستخدام في الكلاس
*/
```