Úkolem je realizovat třídu, která bude reprezentovat datum. Reprezentovaná data budou odpovídat pravidlům Gregoriánského kalendáře, je požadováno, aby šlo reprezentovat data nejméně v rozmezí let 2000 až 2030.

Třída `CDate` musí splňovat následující rozhraní:

- konstruktor s parametry (y,m,d) vytvoří novou instanci třídy s datem nastaveným podle předaných hodnot. Konstruktor musí kontrolovat, zda zadané datum je platné. Pokud platné není, musí vyvolat výjimku `InvalidDateException`,
- operátorem + lze k instanci `CDate` přičíst celé číslo a tím se posunout vpřed o zadaný počet dní (vzad pro záporné číslo),
- operátorem - lze od instance `CDate` odečíst celé číslo a tím se posunout vzad o zadaný počet dní (vpřed pro záporné číslo),
- operátorem - lze od sebe odečíst dvě instance `CDate` a tím získat počet dní mezi těmito dvěma daty,
- operátory ++ a -- v prefixové a v postfixové notaci lze zvyšovat/snižovat datum o 1 den, operátory mají obvyklé chování,
- operátory ==, !=, >, >=, < a <= lze porovnávat dvojici instancí `CDate`, v této relaci budou data v budoucnosti větší než data v minulosti,
- operátorem << lze zobrazit instanci `CDate` v zadaném streamu. Při zobrazování se používá ISO formát (%Y-%m-%d, tedy např. 2000-01-31). V povinných testech bude vždy použit tento implicitní formát. Bonusové testy požadují implementaci manipulátoru `date_format`, kterým lze formát řídit.
- operátorem >> lze přečíst instanci `CDate` ze zadaného streamu. V povinných testech je na vstupu očekáváno datum v ISO formátu %Y-%m-%d. Pokud se nepodaří datum načíst (formát, neplatné datum, ...), operátor zajistí nastavení fail bitu a ponechá původní obsah instance `CDate`. Stejně jako výstupní operátor, i vstup lze řídit pomocí manipulátoru `date_format`, tato možnost je požadovaná v bonusovém testu.

```
#ifndef __PROGTEST__
#include <cstdio>
#include <ctime>
#include <cassert>
#include <iostream>
#include <iomanip>
#include <string>
#include <sstream>
using namespace std;
#endif /* __PROGTEST__ */

class InvalidDateException
{
};

//=================================================================================
==================
// date_format manipulator - a dummy implementation. Keep this code unless you
implement your
// own working manipulator.
ios_base & dummy_date_format_manipulator ( ios_base & x )
{
  return x;
```

```
}

ios_base & ( * ( date_format ( const char * fmt ) ) ) ( ios_base & x )
{
  return dummy_date_format_manipulator;
}
//================================================================================
==================
class CDate
{
  public:
    // constructor ( y,m,d )
    // operator + int
    // operator - int
    // operator - CDate
    // operators ++, --, both prefix and postfix
    // operators ==, !=, <=, <, >=, >
    // operators << and >>
};
```

Odevzdávejte soubor, který obsahuje implementovanou třídu `CDate` a případné další podpůrné třídy a funkce. Třída musí splňovat popsané veřejné rozhraní - pokud Vámi odevzdané řešení nebude obsahovat popsané rozhraní, dojde k chybě při kompilaci. Do třídy si ale můžete doplnit další metody (veřejné nebo i privátní) a členské proměnné. Odevzdávaný soubor musí obsahovat jak deklaraci třídy (popis rozhraní) tak i definice metod, konstruktoru a destruktoru. Je jedno, zda jsou metody implementované inline nebo odděleně. Odevzdávaný soubor nesmí obsahovat vkládání hlavičkových souborů a funkci `main` (funkce `main` a vkládání hlavičkových souborů může zůstat, ale pouze obalené direktivami podmíněného překladu jako v ukázce níže).

Tato úloha není zaměřená na rychlost. Vzhledem k rozsahu testovaných hodnot lze při troše obezřetnosti použít knihovní volání pro konverze data (`ctime`).

Úloha má povinnou a bonusovou část. V povinné části není použit manipulátor `date_format`, tedy je testovaný pouze výchozí formát data (%Y-%m-%d). V bonusové části je navíc požadováno, aby tento manipulátor správně fungoval. Pokud se rozhodnete bonusovou část vynechat, ponechte ve Vašem zdrojovém kódu dodanou "dummy" implementaci manipulátoru (jinak kód nepůjde přeložit).

Bonusové testy používají Vámi dodaný manipulátor `date_format`, který pomocí řetězce popisuje požadované konverze. Formát může obsahovat:

- znaky kromě znaků procento - na vstupu je takový znak na daném místě očekáván a je přeskočen. Není-li na vstupu znak nalezen, je to považováno za chybu. U výstupní konverze je takový znak přidán k výstupu. V ISO formátu %Y-%m-%d jsou takovým znakem pomlčky.
- Znak procento uvozuje den (%d), měsíc (%m) nebo rok (%Y), který se na daném místě vyskytuje v řetězci data. Pro vstup se očekává celé číslo určující den/měsíc/rok, pro výstupní konverzi to znamená, že se k výstupu připojí desítková hodnota dne/měsíce/roku, zleva doplněná nulami.
- Znak procento následovaný znakem jiným než Y, m nebo d, (např. %x, %%, ...) udává, že na

daném místě řetězce data se vyskytuje/má zobrazit takový znak (tedy v příkladu x, %, ...). Tedy ISO formát data by šel s takovou konvencí ekvivalentně přepsat jako %Y%-%m%-%d.

Je-li manipulátor `date_format` jednou zaslaný do streamu, platí tento formát pro všechny další konverze až do případné další změny formátu (tedy chová se podobně jako manipulátory `hex`, ...). Při implementaci mějte na paměti, že v každém streamu může být nastaven jiný manipulátor.

Formát na výstupu může být libovolný (extrém je formát "hello kitty" v ukázce). Vstupní formát má omezení, aby šlo vstup jednoznačně zpracovat: den, měsíc a rok (%d, %m a %Y) musí být ve formátu uveden a to právě jednou. V ukázce je vidět, že vstupní formáty s chybějící či nadbytečnou konverzí vedou k chybě při následném načítání.

Ukázka použití třídy:

```
ostringstream oss;
istringstream iss;

CDate a ( 2000, 1, 2 );
CDate b ( 2010, 2, 3 );
CDate c ( 2004, 2, 10 );
oss . str ("");
oss << a;
assert ( oss . str () == "2000-01-02" );
oss . str ("");
oss << b;
assert ( oss . str () == "2010-02-03" );
oss . str ("");
oss << c;
assert ( oss . str () == "2004-02-10" );
a = a + 1500;
oss . str ("");
oss << a;
assert ( oss . str () == "2004-02-10" );
b = b - 2000;
oss . str ("");
oss << b;
assert ( oss . str () == "2004-08-13" );
assert ( b - a == 185 );
assert ( ( b == a ) == false );
assert ( ( b != a ) == true );
assert ( ( b <= a ) == false );
assert ( ( b < a ) == false );
assert ( ( b >= a ) == true );
assert ( ( b > a ) == true );
assert ( ( c == a ) == true );
assert ( ( c != a ) == false );
assert ( ( c <= a ) == true );
assert ( ( c < a ) == false );
assert ( ( c >= a ) == true );
assert ( ( c > a ) == false );
a = ++c;
oss . str ( "" );
oss << a << " " << c;
assert ( oss . str () == "2004-02-11 2004-02-11" );
a = --c;
oss . str ( "" );
oss << a << " " << c;
assert ( oss . str () == "2004-02-10 2004-02-10" );
a = c++;
oss . str ( "" );
```

```
oss << a << " " << c;
assert ( oss . str () == "2004-02-10 2004-02-11" );
a = c--;
oss . str ( "" );
oss << a << " " << c;
assert ( oss . str () == "2004-02-11 2004-02-10" );
iss . clear ();
iss . str ( "2015-09-03" );
assert ( ( iss >> a ) );
oss . str ("");
oss << a;
assert ( oss . str () == "2015-09-03" );
a = a + 70;
oss . str ("");
oss << a;
assert ( oss . str () == "2015-11-12" );

CDate d ( 2000, 1, 1 );
try
{
   CDate e ( 2000, 32, 1 );
   assert ( "No exception thrown!" == NULL );
}
catch ( ... )
{
}
iss . clear ();
iss . str ( "2000-12-33" );
assert ( ! ( iss >> d ) );
oss . str ("");
oss << d;
assert ( oss . str () == "2000-01-01" );
iss . clear ();
iss . str ( "2000-11-31" );
assert ( ! ( iss >> d ) );
oss . str ("");
oss << d;
assert ( oss . str () == "2000-01-01" );
iss . clear ();
iss . str ( "2000-02-29" );
assert ( ( iss >> d ) );
oss . str ("");
oss << d;
assert ( oss . str () == "2000-02-29" );
iss . clear ();
iss . str ( "2001-02-29" );
assert ( ! ( iss >> d ) );
oss . str ("");
oss << d;
assert ( oss . str () == "2000-02-29" );

//=============================================================================
// bonus tests
//=============================================================================

CDate f ( 2000, 5, 12 );
oss . str ("");
oss << f;
assert ( oss . str () == "2000-05-12" );
oss . str ("");
oss << date_format ( "%Y/%m/%d" ) << f;
assert ( oss . str () == "2000/05/12" );
oss . str ("");
oss << date_format ( "%d.%m.%Y" ) << f;
```

```
assert ( oss . str () == "12.05.2000" );
oss . str ("");
oss << date_format ( "%m/%d/%Y" ) << f;
assert ( oss . str () == "05/12/2000" );
oss . str ("");
oss << date_format ( "%Y%m%d" ) << f;
assert ( oss . str () == "20000512" );
oss . str ("");
oss << date_format ( "hello kitty" ) << f;
assert ( oss . str () == "hello kitty" );
oss . str ("");
oss << date_format ( "%d%d%d%d%d%d%m%m%m%Y%Y%Y%%%%%%%%%%" ) << f;
assert ( oss . str () == "121212121212050505200020002000%%%%%" );
oss . str ("");
oss << date_format ( "%Y-%m-%d" ) << f;
assert ( oss . str () == "2000-05-12" );
iss . clear ();
iss . str ( "2001-01-02" );
assert ( ( iss >> date_format ( "%Y-%m-%d" ) >> f ) );
oss . str ("");
oss << f;
assert ( oss . str () == "2001-01-02" );
iss . clear ();
iss . str ( "05.06.2003" );
assert ( ( iss >> date_format ( "%d.%m.%Y" ) >> f ) );
oss . str ("");
oss << f;
assert ( oss . str () == "2003-06-05" );
iss . clear ();
iss . str ( "07/08/2004" );
assert ( ( iss >> date_format ( "%m/%d/%Y" ) >> f ) );
oss . str ("");
oss << f;
assert ( oss . str () == "2004-07-08" );
iss . clear ();
iss . str ( "2002*03*04" );
assert ( ( iss >> date_format ( "%Y*%m*%d" ) >> f ) );
oss . str ("");
oss << f;
assert ( oss . str () == "2002-03-04" );
iss . clear ();
iss . str ( "C++09format10PA22006rulez" );
assert ( ( iss >> date_format ( "C++%mformat%dPA2%Yrulez" ) >> f ) );
oss . str ("");
oss << f;
assert ( oss . str () == "2006-09-10" );
iss . clear ();
iss . str ( "%12%13%2010%" );
assert ( ( iss >> date_format ( "%%%m%%%d%%%Y%%" ) >> f ) );
oss . str ("");
oss << f;
assert ( oss . str () == "2010-12-13" );

CDate g ( 2000, 6, 8 );
iss . clear ();
iss . str ( "2001-11-33" );
assert ( ! ( iss >> date_format ( "%Y-%m-%d" ) >> g ) );
oss . str ("");
oss << g;
assert ( oss . str () == "2000-06-08" );
iss . clear ();
iss . str ( "29.02.2003" );
assert ( ! ( iss >> date_format ( "%d.%m.%Y" ) >> g ) );
oss . str ("");
```

```
oss << g;
assert ( oss . str () == "2000-06-08" );
iss . clear ();
iss . str ( "14/02/2004" );
assert ( ! ( iss >> date_format ( "%m/%d/%Y" ) >> g ) );
oss . str ("");
oss << g;
assert ( oss . str () == "2000-06-08" );
iss . clear ();
iss . str ( "2002-03" );
assert ( ! ( iss >> date_format ( "%Y-%m" ) >> g ) );
oss . str ("");
oss << g;
assert ( oss . str () == "2000-06-08" );
iss . clear ();
iss . str ( "hello kitty" );
assert ( ! ( iss >> date_format ( "hello kitty" ) >> g ) );
oss . str ("");
oss << g;
assert ( oss . str () == "2000-06-08" );
iss . clear ();
iss . str ( "2005-07-12-07" );
assert ( ! ( iss >> date_format ( "%Y-%m-%d-%m" ) >> g ) );
oss . str ("");
oss << g;
assert ( oss . str () == "2000-06-08" );
```

Nápověda:

- Podle zvolené implementace může hrát roli, že některé dny nemají 24 hodin.
- Pokud budete implementovat manipulátor, nastudujte si metody `xalloc`, `register_callback`, `pword` a `iword` ve třídě `ios_base`.