The task is to develop a C++ class `CDate` representing a date. The class simplifies the processing of dates, in particular, it allows data increment/decrements, output, and validation. The interface of the class is:

a constructor (y, m, d)
: the constructor fills the instance with the date from the parameters. If the date is invalid (see below), the constructor throws `InvalidDateException`.

copy constructor, operator =
: the constructor/op = will be implemented if your internal representation requires it,

destructor
: destructor will be present if your internal representation requires it,

operator <<
: will output the date in a textual form to the given output stream. The required format of the date is `YYYY-MM-DD`, i.e. the day of month and the month is printed as two decimals (with the leading zero if necessary), and the year is printed without any further formatting (the valid years are at least 1600 (see below), thus the year is always at least 4 decimal digits).

operators +/-
: derive a new `CDate` instance where the date is shifted accordingly. The shift is given by the right hand side operand - number of days, months, years, or any combination of them. In general, the change may lead to an invalid date. In such a case, the operators shall throw `InvalidDateException`. The processing is intuitive:
  - years are added/subtracted from the year in the left hand operand. This may lead to an invalid date, e.g. valid date 2000-02-29 incremented by 1 year results in 2001-02-29 (an invalid date). Next, invalid date may be derived if the year component decreases below 1600. Either invalid situation must be handled, by the +/- operator, `InvalidDateException` must be thrown if the resulting date is invalid.
  - months are added/subtracted from the months in the left hand operand and possibly even change the year in the left hand operand. Again, this may lead to an invalid date, e.g. valid date 2018-03-31 incremented by one month results in an invalid date 2018-04-31. Next, invalid date may be derived if the year component decreases below 1600. The operators must throw `InvalidDateException` in these cases.
  - days are added/subtracted to the day component of the date in the left hand operand. The operation may change the month and year component as well if the number of added/subtracted days is high. The add/subtract operation itself does not throw any exceptions (invalid date cannot be derived this way, e.g. 2018-04-30 + 2 days results in valid date 2018-05-02), the only problem may arise if the year component decreases below 1600 (which is considered invalid date in this homework, thus this is the only case where the exception may be actually thrown here).

  The day/month/year components may be combined in the add/subtract operations. The validity of the date must be checked after each operation, if the date is invalid at any time in the evaluation, the exception must be thrown.

operator -
: another overload of operator - allows the subtraction of two `CDate` instances, the result is an integer that represents the number of days elapsed between the two dates.

operators ==, !=, and <
: compare the dates in the natural way,

operator +=

updates the left-hand operand, the right hand operand(s) and the processing is the same as in the case of operator +.

Submit a source file with your implementation of `CDate` class. The submitted file shall not contain any #include directives nor `main` function. If your `main` function or #include remains in the file, please place them into a conditional compile block.

This task does not provide the required class interface. Instead, you are expected to develop the interface yourself. Use the description above, the attached examples, and your knowledge of overloaded operators.

There are bonus tests included in this homework, these tests try the following:

- the shifts may be written in the form of numeric literals with suffixes `_day`, `_days`, `_month`, `_months`, `_year`, and `_years`. The notation is clear from the attached sample code, see the second half of `main`. The suffix notation may be introduced in C++11, the standard offers this syntax by means of overloaded suffix operators. If you decide to submit your solution with this extension, uncomment `#define TEST_LITERALS` directive in the submitted code. If the directive is not defined, the overloaded suffix operator tests is disabled completely and is evaluated 0 points. On the other hand, if you decide not to implement the overloaded suffix operators, then keep the directive `#define TEST_LITERALS` commented out.
- The first bonus test further tests the robustness of your output operator. An ideal implementation would provide correct results in the form YYYY-MM-DD, regardless of the actual stream settings. Moreover, the implementation would restore the settings of the stream when it finishes. There is an example tests in the attached source at the end of the extended test block.
- The second bonus tests the efficiency of operators +, -, +=, and the efficiency of the date difference. A naive solution that iterates over all days is too slow to pass the tests. The test uses dates in far future, e.g. years exceeding 100000.

The common Gregorian calendar is used through this homework. All dates before year 1600 are considered invalid, the first valid date is 1600-01-01 (thus you do not have to care about the shift from Julian to Gregorian calendar). The months are either 30 or 31 days, the only exception is February which is either 28 (standard year) or 29 days (a leap year). The Gregorian calendar rules state:

1. in general, years are standard years,
2. exception of which are years divisible by 4 (that are leap years),
3. exception of which are years divisible by 100 (that are standard years),
4. exception of which are years divisible by 400 (that are leap years),
5. exception of which are years divisible by 4000 (that are standard years). Technically, the last rule has not been officially adopted yet, but the rule is under consideration. Nevertheless, will assume the rule for the purpose of this homework.

Thus years 1601, 1602, 1603, ... ,2001, 2002, 2003, 2005, ... are standard years (rule #1), years 1604, 1608, ..., 2004, 2008, ..., 2096, 2104, ... are leap years (rule #2), years 1700, 1800, 1900, 2100, 2200,

2300, ... are standard years (rule #3), years 1600, 2000, 2400, ..., 3600, 4400, ... are leap years (rule #4), and years 4000, 8000, ... are standard years (rule #5).

The homework assumes that valid dates are dates that follow the Gregorian calendar, starting from date 1600-01-01. If there is created a date that does not follow the Gregorian rules, or there is created a date before 1600-01-01 (either during the construction of `CDate` instance, as a result of addition/subtraction, or even in the middle of the addition/subtraction computation), the constructor/overloaded operator must detect the invalid date and throw `InvalidDateException`. The declaration of the exception is included in the testing environment, the following statement may be used to throw the exception:

```
throw InvalidDateException();
```

Please note that the exception is thrown by-value (i.e. we do not create a dynamically allocated exception via `new InvalidDateException();`). This is consistent with the exception handler which accepts a value reference (it does not accept a pointer).

Advice

- The testing environment uses output operator (<<) to examine your instance. If your overloaded operator `<<` does not work properly, the tests will be negative.
- Implement the output operator properly -- do not blindly send the data to `cout`. Instead, send the data to the output stream passed as the parameter. Do not add any extra whitespace/newline characters.
- If your program does not compile (and especially if it compiles locally, however it does not compile in Progtest), there might be some problem in your interface design. Check your operator overloads, pay special attention to the `const` qualifiers.
- Operator += is a bit tricky. The problem is in the right-hand side operand. The operator must detect invalid date even in the middle of the processing of the right-hand side operator list. The design of the solution and the design of the classes is tightly connected with this behavior. Do not start your implementation until you solve this problem.
- STL classes `std::vector`, `std::list`, and `std::string` are available. However, the rest of STL is not.
- The built-in time functions `mktime` and `localtime` are not particularly good for this homework: the range of valid dates might be unacceptable (e.g. dates before 1970-01-01 cause problems on 32 bit systems), the functions are not easy-to-use, their behavior is influenced by timezone and daylight saving time settings, and the functions may use different algorithm for leap years (e.g. year 4000 may be considered a leap year by the functions).
- The singular and plural suffixes in the bonus test are not distinguished. That is, literals `1_day` and `1_days` are both considered valid and equivalent. The same applies e.g. for `12_months` and `12_month` - they are also both correct and equivalent.