## Submission: One word document containing all your code. Due before the next class.

## Goal

Practice using the following:

* Simple file documentation
* Using header and source files
* Constructors and destructors
* Overloading the + and the << operators

# Requirements

Implement a Date class using a C++ program. Match the output against the sample output given the provided test harness.

The Date Class

The members must be declared in a header file and implemented in a source file. You may also have another source file containing the test harness.

|  |
| --- |
| **Date**  Class |
| **Fields** |
| - year\_ : **int** = 2019  - month\_ : **int** = 1  - day\_ : **int** = 1 |
| **Methods** |
| + «constructor» Date(  year : **int**,  month : **int**,  day : **int**)  + «destructor» ~Date()  + **operator**+(rhs : **const Date**&) : **Date**  + **operator**+(rhs : **const int**) : **Date**  + «**friend**» **operator**<<(  os : **ostream**&,  rhs : **const Date**&) : **ostream**&  - normalize() : **void** |

## Description of class members

### Fields

There are three int fields that are private.

year\_ – this is a private int field that represents the year value of this object.

month\_ – this is a private int field that represents the month value of this object.

day\_ – this is a private int field that represents the day value of this object.

### Constructors

**Date(int year, int month, int day)** – This public constructor takes three int arguments and assigns them to the appropriate fields. Please be aware of the default argument declared in the header file.

~~Before assignments it does the following sanity checks:~~

* ~~If the argument year is less than 0, then the field is set to 2022.~~
* ~~If the argument month is not within the range 1 to 12, then the field is set to 1.~~
* ~~If the argument day is not within the range 1 to 30, then the field is set to 1.~~

### Destructors

**~Date()** – This public destructor does not implement any logic at this time. It simply output the string “destructor” to the console.

### Methods

There are three friend methods that overload the plus and the insertion operators.

Date operator+(const Date& rhs) – This method overloads the plus operator. It facilitates the addition of two Date objects using the arithmetic + operator. It returns a Date object with the following features:

* year\_ field as the sum of the year\_ fields of the argument and the appropriate field.
* month\_ field as the sum of the month\_ fields of the argument and the appropriate field.
* day\_ field as the sum of the day\_ fields of the argument and the appropriate field.

Date operator+(const int rhs) – This method overloads the plus operator. It facilitates the addition of two Date objects using the arithmetic + operator. It returns a Date object with the following features:

* year\_ field the same as the year\_ field.
* month\_ field as the same as of the month\_ field.
* day\_ field as the sum of the day\_ field and the argument.

std::ostream& operator<<(std::ostream& os, const Date& date) – This method overloads the insertion operator. It facilitates the easy display of an object using and stream object. See the sample output for clues on how to implement the logic.

void Date::normalize()– This private method constrains the day\_ and the month\_ fields to a sensible range. (You may assume that day\_ may range from 1 to 30 and month\_ may range from 1 to 12.)

This method is call whenever addition is performed.

### Test harness

Copy the following statements and paste into your main method.

std::cout << "\ncreating an anonymous object - 1\n";

std::cout << Week01::Date(2019) << '\n';

std::cout << "\ncreating a named object - 2\n";

Week01::Date d1(2020, 3, 28);

std::cout << d1 << '\n';

std::cout << "\ncreating another named object - 3\n";

Week01::Date d2(2, 10, 14);

std::cout << d2 << '\n';

std::cout << "\ncreating another named object - 4\n";

Week01::Date d3 = d1 + d2;

std::cout << d1 << " + " << d2 << " = " << d3 << '\n';

int days = 84;

//setting a new object to d3

std::cout << "\nsetting a new object to an existing variable\n";

d3 = d2 + days;

std::cout << d2 << " + " << days << " = " << d3 << '\n';

std::cout << d1 << '\n';

std::cout << d2 << '\n';

std::cout << d3 << '\n';

### Sample output

Try to understand why and when constructors and destructor are called.

creating an anonymous object - 1

constructor

<2019-1-1>

destructor

creating a named object - 2

constructor

<2020-3-28>

creating another named object - 3

constructor

<2-10-14>

creating another named object - 4

constructor

destructor

<2020-3-28> + <2-10-14> = <2023-2-12>

setting a new object to an existing variable

constructor

destructor

destructor

<2-10-14> + 84 = <3-1-8>

<2020-3-28>

<2-10-14>

<3-1-8>

destructor

destructor

destructor