

# Date Class

Stephen Gerkin

October 16, 2019

CIST 2362 CRN 22282

## Program Description

A date class is to be constructed to represent a standard date within the Gregorian calendar. The class functions should then be demonstrated to illustrate how the class functions. The class should store a year, month, and a date and provide getters and setters for each value. The class should be able to print the date to the screen in a variety of formats. Additional program requirements listed below.

## Program Requirements

- The date should be displayable in the following formats:  
MM/DD/YYYY – 12/25/2018  
Month DD, YYYY – December 25, 2018  
DD Month YYYY – 25 December 2018
- The class should have postfix and prefix increment operators.
- The class should have postfix and prefix decrement operators.
- The class should have a subtraction operator for determining the number of days between two dates.
- The class should have an insertion operator override for use with *cout*.
- The class should have an extraction operator override for use with *cin*.
- When a date is incremented or decremented, the date should change to represent the previous or next day.
- When a date is incremented at the end of the month, the following month should be set. For example, on July 31, the date should be incremented to August 1.
- When a date is incremented at the end of the year, the date should change to the following year and reset to January 1.
- When a date is decremented at the beginning of the month, the date should change to last day of the previous month. For example, on July 1, the date should decrement to June 30.
- When a date is decremented at the beginning of the year, the date should change to December 31<sup>st</sup> of the previous year.

## Input Validation

- Do not accept values for the day greater than 31.
- Do not accept values for the day less than 1.
- Do not accept values for the month greater than 12.
- Do not accept values for the month less than 1.
- The overloaded >> operator should not accept an invalid date.

## Programming Strategy:

The Date class will hold only the responsibility of representing a date in the Gregorian calendar. It will provide getters and setters for accessing or modifying the values. The date class will be responsible for knowing how many days are in each month (i.e. January has 31, June has 30, February has 28 except in leap years where it has 29). The class will be responsible for determining if a year is a leap year or not. The class will be responsible for calculating the number of days between two dates. This will be accomplished by storing two date values. The date that comes first chronologically will be incremented until it is equal to the second date. The amount of times this increment happens will be used to calculate the number of days between the two dates.

## Date Class Documentation

Date class encapsulates and represents a date in the Gregorian calendar.

Inputs are assumed to be valid dates within Gregorian calendar (meaning any date entered before October 1582 should be considered only an estimate when calculating number of days between dates when before this point in time).

Any year entry prior to 0 CE (1 BCE or prior) is not supported by this class. Hypothetically, this class should be able to support any year up to 2147483647, but this is definitely not recommended and has not been tested. Additionally, attempting to calculate the date range between the beginning of the Common Era and the year 2147483647 will take a considerable amount of processor time as the time complexity of computing this is  $O(n)$ .

Except when using the istream extraction operator ( `>>` ), all Date values should be entered in descending order of magnitude (YYYY/MM/DD) to follow standard programming protocols for datetime recording.

The exclusion for the `>>` operator is to allow user or file input given a specific formatting style. For instance, if the user is accustomed to entering dates in the form MM/DD/YYYY, this can be accomplished by setting the format using the DATE\_STRING\_FORMAT enumerator to US\_SHORT or US\_LONG.

The class provides several methods for modifying the values of a date and methods for calculating the differences between dates. Further information about these methods can be found internally documented with each function definition or with each class method description in the external documentation.

### *Exception Safety:*

Methods that are exception safe are explicitly marked `noexcept`. All constructors (except copy constructor) and methods to change the date values can throw `std::out_of_range` exceptions for invalid dates values.

For more detail, see each individual method description.

### *Thread Safety:*

To modify or change the existing values of a Date object, one must change all three values at method call or instantiation. However, the current string format option and the list of valid delimiters can be changed independently of changing the values of a date. Therefore, it is wise to consider this class NOT

thread safe overall but can be considered probably safe if no modifications are to be done after initialization of formatting choice and delimiters.

Thread safety has not been tested and this class is not designed to fulfill that purpose.

## Member Variables

Date Type	Name	Description
int	year	For storing a year.
int	month	For storing a month value.
int	day	For storing a day value.
DATE_STRING_FORMAT	dateStringFormat	For storing a format preference for displaying dates.
std::set<char>	validDelimiters	For storing valid delimiters between date values.

## Member Enumeration

*Date::DATE\_STRING\_FORMAT*

For determining string formatting of dates to be displayed.

*Enumerator:*

US_SHORT	Example: 07/18/2019
US_LONG	Example: July 18, 2019
EU_SHORT	Example: 18/07/2019
EU_LONG	Example: 18 July 2019
LAST	For iterating over the enumerator. Do not set any values to this.

## Constructors and Destructors

*Date::Date ()[noexcept]*

No argument constructor creates an object with the current local **Date**.

This is accomplished using the **now()** static method to get the system clock. The default formatting style is set to US\_SHORT.

*Date::Date (int year, int month, int day, Date::DATE\_STRING\_FORMAT format)*

Full argument constructor.

Consider default. Takes integer values for a **Date** object to construct. Values should be passed in descending order of magnitude (YY/MM/DD) to avoid confusion between various date formatting options. Sets the string formatting for the date by provided parameter.

*Parameters*

<i>year</i>	int value for year.
<i>month</i>	int value for month, must be between 1 and 12.

<i>day</i>	int value for day, must be between 1 and max days in month.
<i>format</i>	the specified format in which the string representation of <b>Date</b> should be created.

#### Exceptions

<i>std::out_of_range</i>	exception if month is not valid or day is not valid given a specified month or negative value entered for year.
--------------------------	---

*Date::Date (int year, int month, int day)*

Constructor without DATE\_STRING\_FORMAT specified.

Sets the format to US\_SHORT by default. Takes integer values for a **Date** object to construct. Values should be passed in descending order of magnitude (YY/MM/DD) to avoid confusion between various date formatting options.

#### Parameters

<i>year</i>	int value for year.
<i>month</i>	int value for month, must be between 1 and 12.
<i>day</i>	int value for day, must be between 1 and max days in month.

#### Exceptions

<i>std::out_of_range</i>	exception if month is not valid or day is not valid given a specified month.
--------------------------	--

*Date::Date (const Date & obj)[noexcept]*

Copy constructor.

Creates a deep copy.

#### Parameters

<i>obj</i>	<b>Date</b> object to copy.
------------	-----------------------------

*Date::~~Date ()[default]*

Default destructor.

## Member Functions

*bool Date::addDelimiter (char delimiter)[noexcept]*

Adds a character to the set of delimiters.

Returns true if delimiter was not already in the set and was added. Returns false if the delimiter was already in the set.

#### Parameters

<i>delimiter</i>	The character we want to add to the set.
------------------	--

#### Returns

True if successfully inserted.

*int Date::daysBetween (const Date & date1, const Date & date2)[static], [noexcept]*

Calculates the days between two supplied dates.

Order of dates entered does not matter.

#### Parameters

<i>date1</i>	First <b>Date</b> to use for calculation.
<i>date2</i>	Second <b>Date</b> to use for calculation.

#### Returns

Integer number of days between the two supplied dates.

*int Date::daysBetween (const Date & dateToCheck)[noexcept]*

Calculates the days between \*this **Date** and the supplied **Date** value.

#### Parameters

<i>dateToCheck</i>	The <b>Date</b> value to check for range.
--------------------	---

#### Returns

An integer of the number of days between \*this **Date** and supplied **Date**.

*int Date::daysInMonth (int monthNum)*

Determines the max number of days given a month and this->year.

For example, given April, June, September, or November, return 30. For February, will check if it is a leap year or not to determine value. All other months have 31 days.

#### Parameters

<i>month</i>	The month number value to get the max number of days for.
--------------	---

#### Returns

The maximum number of days in a given month.

*int Date::daysInMonth (int year, int month)[static]*

Determines the max number of days given a month and a year.

For example, given April, June, September, or November, return 30. For February, will check if it is a leap year or not to determine value. All other months have 31 days.

#### Parameters

<i>year</i>	The year of the month to check. Required for leap year checks.
<i>month</i>	The month number value to get the max number of days for.

#### Returns

The maximum number of days in a given month.

*bool Date::equalDateValues (const Date & dateToCheck) const[noexcept]*

Determines if \*this **Date** object is equal to another **Date** object.

Warning: Does not check if both dates use the same format for string formatting! This only validates the date values.

#### Parameters

<i>dateToCheck</i>	The date to check against.
--------------------	----------------------------

#### Returns

True if *\*this* **Date** is equal to the *dateToCheck*.

*Date::DATE\_STRING\_FORMAT Date::getStringFormat () const[noexcept]*

Getter for *this->dateStringFormat*.

*std::string Date::getStringFormatDisplayString () const[noexcept]*

Get a string for displaying what the current date format is.

#### Returns

A string representation of *this->dateStringFormat*.

*int Date::getDay () const[noexcept]*

Getter for *this->day*.

*int Date::getMonth () const[noexcept]*

Getter for *this->month*.

*std::string Date::getMonthName () const[noexcept]*

Determines the string representation for *this->month*.

Unlike the static class method that determines a month name with a parameter, this will never throw an exception because *this->month* must be a valid month number in order for *\*this* to exist.

#### Returns

A string representation of *this->month*.

*std::string Date::getMonthName (int month)[static]*

Returns a string representation for a given month number.

#### Parameters

<i>month</i>	int value to find the name of a month for.
--------------	--

#### Returns

A string representation of the month number

#### Exceptions

<i>std::out_of_range</i>	exception if month is not valid
--------------------------	---------------------------------

*const std::set< char > & Date::getValidDelimiters () const[noexcept]*

Getter for *this->validDelimiters*.

*int Date::getYear () const[noexcept]*

Getter for *this->year*.

*bool Date::isBefore (const Date & checkDate)[noexcept]*

Determines if this **Date** object comes before provided date object.

#### Parameters

<i>checkDate</i>	The date to check against this.
------------------	---------------------------------

#### Returns

True if this **Date** comes before the checkDate.

*bool Date::isBefore (const Date & date1, const Date & date2)[static], [noexcept]*

Determines if date 1 comes before date 2.

Uses short circuit OR checks to exit early based on descending magnitude. For instance, if year 1 is less than year 2, we know date1 is before date2. But if they are the same year, we then check month, and finally day. If all checks evaluate as false, we know date1 is not before date2.

#### Parameters

<i>date1</i>	The date to check if before.
<i>date2</i>	The date to check against the before.

#### Returns

True if date1 is before date2.

*bool Date::isLeapYear ()[noexcept]*

Determines if this->year is a leap year.

#### Returns

True if this->year is a leap year.

*bool Date::isLeapYear (int year)[static]*

Checks if a year is a leap year (29 days in February).

#### Parameters

<i>year</i>	The year to check.
-------------	--------------------

#### Returns

True if the year is a leap year.

*Date & Date::nextDay ()[noexcept]*

Increments the current date held by this by one day and returns a pointer back to this.

#### Returns

this after incrementing.

*Date Date::now ()[static]*

Retrieves the current time from the system clock and sets **Date** values to the current date.

Uses local time and default format style.

#### Returns

**Date** object encapsulating the current local system clock time.

*Date Date::now (DATE\_STRING\_FORMAT format)[static]*

Retrieves the current time from the system clock and sets **Date** values to the current date.

Uses local time for determining date and selected format style.

#### Parameters

<i>format</i>	DATE_STRING_FORMAT type for formatting Strings of <b>Date</b>
---------------	---

#### Returns

**Date** object encapsulating the current local system clock time with selected formatting.

*Date & Date::operator++ ()[noexcept]*

Increment Prefix operator overload.

#### Returns

\*this after being decremented.

*Date Date::operator++ (int )[noexcept]*

Increment postfix operator overload.

#### Returns

A temp date holding what \*this is before incrementing.

*int Date::operator- (const Date & subtractDate)[noexcept]*

Returns the days between this **Date** object and the supplied **Date** object.

If the supplied date is after \*this date, will return a negative value.

#### Parameters

<i>subtractDate</i>	The date to subtract from *this <b>Date</b> .
---------------------	---

#### Returns

The number of days between \*this **Date** and the supplied date.

*Date & Date::operator-- ()[noexcept]*

Decrement Prefix operator overload.

#### Returns

\*this after being decremented.

*Date Date::operator-- (int )[noexcept]*

Decrement Postfix operator overload.

#### Returns

A temp date holding what \*this is before decrementing.

*bool Date::operator== (const Date & dateToCheck) const[noexcept]*

Operator overload for checking equality.

Unlike equalValues method, this WILL validate that string format selection is the same between objects!

#### Parameters

<i>dateToCheck</i>	The date to check against.
--------------------	----------------------------

#### Returns

True if \*this date is equal to the dateToCheck.

*Date & Date::previousDay ()[noexcept]*

Decrements the current date held by this by one day and returns a pointer back to this.

### Returns

this after decrementing.

*bool Date::removeDelimiter (char delimiter)[noexcept]*

Removes a character from the list of valid delimiters.

Does not allow removal of delimiters if there is only one in the set. If the character is not in the list or cannot be erased, this method will return False.

### Parameters

<i>delimiter</i>	The delimiter we want to remove.
------------------	----------------------------------

### Returns

True if the character was removed successfully.

*void Date::setDate (int yyyy, int mm, int dd)*

Setter to change/set a new value for the date.

Values should be passed in descending order of magnitude (YY/MM/DD) to avoid confusion between various date formatting options.

If any field is invalid, the values for year and month will not be changed from original values. For example, let's say \*this **Date** is 07/18/2018. The input given to change the date is yyyy = 1999 mm = 03 dd = 32 This will throw an out\_of\_range exception because day cannot be 32

To make sure we don't now have a date of 03/18/1999, before the exception is thrown, each value is reset to what it was before being changed so the values in \*this will still be year = 2018 month = 07 day = 18

### Parameters

<i>yyyy</i>	int value for year.
<i>mm</i>	int value for month, must be between 1 and 12.
<i>dd</i>	int value for day, must be between 1 and max days in month.

### Exceptions

<i>std::out_of_range</i>	exception if month is not valid or day is not valid given a specified month or year is less than 0.
--------------------------	---

*void Date::setDateStringFormat (Date::DATE\_STRING\_FORMAT newFormat)*

Setter for this->dateStringFormat.

### Exceptions

<i>std::invalid_argument</i>	if attempt to set to <b>Date::LAST</b> .
------------------------------	--

*void Date::setValidDelimiters (const std::set< char > & delimiters)[noexcept]*

Setter for this->validDelimiters.

*std::string Date::toString () const[noexcept]*

Creates a string representation of \*this **Date** based a format set either during construction or using setDateStringFormat(format).

#### Returns

A string representation of \*this **Date**.

*std::string Date::toString (Date::DATE\_STRING\_FORMAT format) const[noexcept]*

Creates a string representation of \*this **Date** based on the supplied format.

#### Parameters

<i>format</i>	The format we wish to use to format a date.
---------------	---

#### Returns

A string representation of \*this **Date**.

#### Friends and Related Functions

*std::ostream & operator<< (std::ostream & stream, const Date & date)[friend]*

Stream insertion operator overload.

Sends a full length date string to the stream.

#### Parameters

<i>stream</i>	an output stream object to send data to.
<i>date</i>	A date object we wish to stream.

#### Returns

Full length US formatted string representation of the date.

*std::istream& operator>> (std::istream & stream, Date & obj)[friend]*

Stream extraction operator overload.

Takes a string representation of a date and converts it into a **Date** object. Valid formatting for string input is fairly strict and as such should not be used anywhere user input is required. Individual values should instead be obtained and validated independently using the static methods provided by this class to validate days and months.

Valid formats are: US: MM/DD/YYYY EU: DD/MM/YYYY

The date formatting **MUST** be set correctly before using this or it could behave in unexpected ways. Note that the default empty constructor for a **Date** sets the date formatting to the default formatting selection so inputting an EU formatted date string into a freshly initialized **Date** object without setting the date format will most likely cause an exception or behave contrary to design.

This operator is more likely than not going to throw an exception. It should be avoided if possible. Input validation within this operator also violates the single responsibility principle and as such does NOT do it very well.

Input should be validated more rigorously outside of using this operator.

#### Parameters

<i>stream</i>	A stream object containing a date string
<i>obj</i>	The <b>Date</b> object we wish to set

### Returns

A clean stream object.

### Exceptions

<code>std::invalid_argument</code>	If formatting of input is incorrect or the values supplied are invalid.
------------------------------------	---

## Unit Testing

Encapsulated by `test_Date` class.

Constructor initiates all tests and displays to the screen if all tests pass successfully. If any test fails, specific information about what test failed with given input, expected output, and actual output information.

*`bool test_Date::runAll ()[static]`*

Runs all unit tests sequentially. If all tests pass, returns TRUE.

*`bool test_Date::test_ConstructorDoesNotThrowExceptionsForUnusualValidInput ()[static]`*

Verifies that the constructor does not throw exceptions for valid input that is uncommon.

This tests a few leap years and for allowing 29 days. Also checks for January 1st, 0 CE just to verify that the year check allows 0 input.

*`bool test_Date::test_ConstructorThrowsExceptionsForBadInput ()[static]`*

Verifies that the the constructor throws an exception for bad date values.

Examples of a bad date are any day greater than 32, days greater than 30 for months only containing 30 days, etc. Anything BCE is checked.

*`bool test_Date::test_decrementPostfixOperator ()[static]`*

Verifies the postfix decrement operator returns the correct information.

*`bool test_Date::test_decrementPrefixOperator ()[static]`*

Verifies the prefix decrement operator returns the correct information.

*`bool test_Date::test_equalDateValues ()[static]`*

Verifies the method `equalDateValues()` returns the correct information.

*`bool test_Date::test_equalityOperator ()[static]`*

Verifies the equalities operator overload returns the correct information.

*`bool test_Date::test_incrementPostfixOperator ()[static]`*

Verifies the postfix increment operator returns the correct information.

*`bool test_Date::test_incrementPrefixOperator ()[static]`*

Verifies the prefix increment operator returns the correct information.

*`bool test_Date::test_istreamOperatorOverloadWithBadFormatInputThrowsException ()[static]`*

Verifies the istream extraction operator overload functions correctly by throwing exceptions with bad input.

*bool test\_Date::test\_istreamOperatorOverloadWithGoodInput ()[static]*

Verifies the istream extraction operator overload functions correctly with good input.

*bool test\_Date::test\_istreamOperatorOverloadWithInvalidFormatThrowsException ()[static]*

Verifies the istream extraction operator overload functions correctly by throwing exceptions with bad input formatting input.

*bool test\_Date::test\_member\_daysBetween ()[static]*

Verifies that the member method daysBetween() returns the correct information.

*bool test\_Date::test\_member\_daysInMonth ()[static]*

Verifies the member method isLeapYear() returns the correct information.

*bool test\_Date::test\_member\_getMonthName ()[static]*

Verifies the static method getMonthName() returns the correct information.

*bool test\_Date::test\_member\_isBefore ()[static]*

Verifies that the member method isBefore() returns the correct information.

*bool test\_Date::test\_member\_isLeapYear ()[static]*

Verifies the member method isLeapYear() returns the correct information.

*bool test\_Date::test\_nextDay ()[static]*

Verifies the method nextDay() returns the correct information.

*bool test\_Date::test\_previousDay ()[static]*

Verifies the method previousDay() returns the correct information.

*bool test\_Date::test\_static\_daysBetween ()[static]*

Verifies that the static method daysBetween() returns the correct information.

*bool test\_Date::test\_static\_daysInMonth ()[static]*

Verifies the static method daysInMonth() returns the correct information.

*bool test\_Date::test\_static\_getMonthName ()[static]*

Verifies the static method getMonthName() returns the correct information.

*bool test\_Date::test\_static\_isBefore ()[static]*

Verifies that the static method isBefore() returns the correct information.

*bool test\_Date::test\_static\_isLeapYear ()[static]*

Verifies the static method isLeapYear() returns the correct information.

*bool test\_Date::test\_subtractOperator ()[static]*

Verifies that the subtraction override returns the correct information.

## Program Demonstration Functions

*void changeDate (Date & date)*

Allows a user to change a date object by inputting new values for a date.

#### Parameters

<i>date</i>	The date we want to change.
-------------	-----------------------------

*void changeDateFormat (Date & date)*

Displays a menu for changing the current date format of the **Date** object.

The date format will be changed based on selections after this function is finished.

#### Parameters

<i>date</i>	The date we wish to change the format for.
-------------	--

*std::vector<std::string> createValidInputStringDisplays (const Date & date)*

Creates a list of valid input formats for a **Date** object.

#### Parameters

<i>date</i>	The <b>Date</b> the user is currently attempting to change.
-------------	---

#### Returns

A list of valid format string examples for the current **Date** object.

*void demonstrateDateClass ()*

Demonstrates the **Date** class by creating a **Date** object and allowing the user to manipulate it in various ways.

Also allows the user to view the status of the unit tests.

*void demonstrateDaysBetweenDates (const Date & date)*

Demonstrates the function to calculate the number of days between two dates.

The current date we're working with is copied to the first date value to have a starting point but can be changed without changing the actual value inside the Date object given to this function.

#### Parameters

<i>date</i>	The current date we're working with.
-------------	--------------------------------------

*void demonstrateSubtractionOperator (const Date & date)*

Demonstrates the subtraction operator for two dates.

If the first date used is after the second date used, it'll display how many days until the second date. Otherwise it'll display how many days it has been since. The Date object given to this function is not modified.

#### Parameters

<i>date</i>	The current date we're working with.
-------------	--------------------------------------

*void displayChangeDateFormatMenu (Date & date)*

Displays the possible format options and gets user input to change the format to.

#### Parameters

<i>date</i>	The <b>Date</b> object we're working with.
-------------	--

*void displayCurrentDateFormat (const Date & date)*

Displays the current date format of the **Date** object.

### Parameters

<code>date</code>	The <b>Date</b> object we're working with.
-------------------	--

## Input and Output Demonstration

Start screen

```
The current date is: 10/16/2019
The display format is: US Short

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...
>>
```

Change Date

```
Make a selection ...
>> 1

Enter a date value:
>> 07/18/2000

The current date is: 7/18/2000
The display format is: US Short

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...
>> █
```

## Change Date Format

```
Make a selection ...
>> 2

Your current format is: US Short
That looks like this: 7/18/2000

1. Format: US Short looks like 7/18/2000
2. Format: US Long looks like July 18, 2000
3. Format: EU Short looks like 18/7/2000
4. Format: EU Long looks like 18 July 2000
Make a selection ...
>> 2

The current date is: July 18, 2000
The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...
>>
```

### Increment Date Twice

The current date is: July 18, 2000

The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...

>> 3

The current date is: July 19, 2000

The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...

>> 3

The current date is: July 20, 2000

The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...

>> \_

## Decrement Date Twice

The current date is: July 20, 2000

The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...

>> 4

The current date is: July 19, 2000

The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...

>> 4

The current date is: July 18, 2000

The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...

>>

Check days between

```
Do you want to use the current date July 18, 2000?
1. Yes
2. No
Make a selection ...
>> 1

What is the second date?
Enter a date value:
>> 07/18/2019

There are 6939 days between July 18, 2000 and July 18, 2019

The current date is: July 18, 2000
The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...
>> _
```

Check Days Between (Do not use current date)

```
Make a selection ...
>> 5

Do you want to use the current date July 18, 2000?
1. Yes
2. No
Make a selection ...
>> 2

What is the first date?
Enter a date value:
>> 01/01/2018
What is the second date?
Enter a date value:
>> 01/01/2019

There are 365 days between January 1, 2018 and January 1, 2019

The current date is: July 18, 2000
The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...
>> _
```

#### Subtract dates (1)

```
Make a selection ...
>> 6

Do you want to use the current date July 18, 2000?
1. Yes
2. No
Make a selection ...
>> 1

What is the second date?
Enter a date value:
>> 07/18/1999

On July 18, 2000 it has been 366 days since July 18, 1999

The current date is: July 18, 2000
The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...
>> _
```

## Subtract Dates (2)

Make a selection ...

>> 6

Do you want to use the current date July 18, 2000?

1. Yes

2. No

Make a selection ...

>> 1

What is the second date?

Enter a date value:

>> 07/18/2001

On July 18, 2000 it will be 365 days until July 18, 2001

The current date is: July 18, 2000

The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...

>>

## Perform Unit Tests

```
Make a selection ...
>> 7

PERFORMING UNIT TESTS ...

TEST: Constructor throws exceptions for bad input ... ALL PASSED
TEST: Constructor does not throw exceptions for unusual valid input ... ALL PASSED
TEST: static isBefore() ... ALL PASSED
TEST: member isBefore() ... ALL PASSED
TEST: static daysBetween() ... ALL PASSED
TEST: member daysBetween() ... ALL PASSED
TEST: operator overload (-) ... ALL PASSED
TEST: equalDateValues() ... ALL PASSED
TEST: equality operator (==) ... ALL PASSED
TEST: nextDay() ... ALL PASSED
TEST: increment operator (++prefix) ... ALL PASSED
TEST: increment operator (postfix++) ... ALL PASSED
TEST: previousDay() ... ALL PASSED
TEST: decrement operator (postfix--) ... ALL PASSED
TEST: decrement operator (--prefix) ... ALL PASSED
TEST: istream operator overload (>>) GOOD INPUT ... ALL PASSED
TEST: istream operator overload (>>) Bad Format Input throws exception ... ALL PASSED
TEST: istream operator overload (>>) out of range throws exception ... ALL PASSED
TEST: static isLeapYear() ... ALL PASSED
TEST: member isLeapYear() ... ALL PASSED
TEST: static daysInMonth() ... ALL PASSED
TEST: member daysInMonth() ... ALL PASSED
TEST: static daysInMonth() ... ALL PASSED
TEST: member daysInMonth() ... ALL PASSED

***ALL TESTS PASSING***

The current date is: July 18, 2000
The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...
>> _
```

#### Bad Date Input

```
Make a selection ...
>> 1

Enter a date value:
>> 02/29/2019
Sorry, that input was not recognized.
Please make sure your input is in one of these forms:
MM-DD-YYYY      MM.DD.YYYY      MM/DD/YYYY
Enter a date value:
>> 02/28/2019

The current date is: February 28, 2019
The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...
>> _
```

#### Bad menu input

```
Make a selection ...
>> a

That was not a valid selection.
Make a selection ...
>> 10
```

## Leap Year

```
Make a selection ...
>> 1

Enter a date value:
>> 02/29/2008

The current date is: February 29, 2008
The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...
>>
```

## Go to next month increment

```
Make a selection ...
>> 3

The current date is: March 1, 2008
The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...
>> █
```

Go to next year increment

```
The current date is: December 31, 2018
The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...
>> 3

The current date is: January 1, 2019
The display format is: US Long
```

Go to previous year decrement

```
The current date is: January 1, 2019
The display format is: US Long

1. Change Date
2. Change Date Format
3. Increment Date
4. Decrement Date
5. Check days between another date
6. Subtract Dates to determine days between
7. Show the status of unit testing for Date.
8. EXIT

Make a selection ...
>> 4

The current date is: December 31, 2018
The display format is: US Long
```