# Assignment 4

You are going to implement three classes in order to create a calendar system which stores information on people's names and their birthdays. Header files are provided to you. You are also provided with three example mains which should help you evaluate your work. Outputs for the example mains are given in comments in their respective files. You are not to change anything in these header files or example main files; your work should be able to compile and run along with these files. You should send only your cpp files. Your work will be evaluated on Visual Studio 2015.

## date class

This class stores a date where day, month, and year information are stored as integers. Constructors and member functions to be implemented in this class are:

**date():** This is the default constructor; sets all variables to 0.

**date(int, int, int):** The constructor where initial values for day, month, and year are initialized with the parameters.

**void set_day(int);**
**void set_month(int);**
**void set_year(int):** Mutator functions to change the values of day, month, and year variables of a date object.

**int get_day() const;**
**int get_month() const;**
**int get_year() const:** Accessor functions to get the values of day, month, and year variables from outside the class. These member functions have a const at the end of definition in order to prevent them from changing any member variables.

**std::string get_string() const:** Returns the date as string in a human-friendly format, such as "18 March 2017".

**int days_since_0() const:** Returns the number of days passed since the date 0/0/0. You should not concern yourself about dates before this time. This function is defined as private since it is not going to be used outside this class. The pupose of this function is to help simplify the compare function.

Remember to take leap years into consideration. If a year is divisible by 400, it is a leap year. If a year is not divisible by 400 but is divisible by 100, it is not a leap year. If a year is not divisible by 100 but is divisible by 4, it is a leap year. A year is not a leap year if it is not divisiblye by 4. Examples:

1600 -> leap year since divisible by 400
1700 -> not leap year since divisible by 100 but not divisible by 4
1704 -> leap year since divisible by 4 but not divisible by 100

1705 -> not leap year since not divisible by 4

**int compare(const date &) const:** Returns how many days are there between the two dates. The first date is the caller object, and the second date is the parameter. If the first date comes after the second one, the result should be positive, and negative otherwise (except when both dates are same, where the result should be 0). There are examples provided in the example_main file for both positive and negative cases.

const identifier at the end of function definition here means that you are not allowed to change any member variables for the caller object. const identifier that is a part of parameter type means that no member variables of the parameter object should change. In order to guarantee this, C++ prevents you from calling any non-const functions here either. For example, you can call the member function get_day here, since it is also const and guarantees that it will not modify any member variable; but you can't call set_month because it may modify a member variable, thus ruining the compare function's constness.

## bday class

This class stores information about birthdays: it stores the date and it stores the name of the birthday person.

**std::string get_string() const:** Returns the birthday information as string in a human-friendly format, such as "John Doe, 1 January 2000". Try to avoid unnecessary code here by using the appropriate member function you implemented in date class.

Rest of the member functions and constructors are straightforward and similar to the ones in date class, so no further explanation is required.

## calendar class

This class stores a number of birthday information and handles some tasks. It stores birthday information as bday objects, and these objects are stored in an STL list.

**void add(const bday &):** Adds the new birthday information to the list. However, a person should not have two different birthdays at the same time. In order to achieve this, a birthday should be added to the list only if the person whose birthday is being added is not present on the list. Otherwise, no new birthdays should be added, but the pre-existing birthday info should be updated. An example is present in the example main file.

**void remove(std::string):** Removes a person's birthday information from the list. If no such person exists, this function should do nothing.

**void print() const:** Prints all birthday information stored in the list in a human-friendly format. Try to avoid unnecessary code here by using the appropriate member function you implemented in bday class.

**void print_incoming(const date &) const:** Prints birthday information for all people in the list whose birthday takes place in the next 7 days (but not past 7 days). The current date is provided as the parameter.