

# CSC3002: Introduction to Computer Science

## Assignment 1

### Assignment description:

You should write your code for each question in several .cpp and .h file (please name it using the question name, e.g. q1.cpp). Please pack your **whole project into a single .zip file**, name it using your **student ID** (e.g. if your student ID is 123456, then the file should be named as 123456.zip), and then submit the .zip file via BB.

For this assignment, you **don't have to** use the sample project. You can create an empty project with QT Creator to start programming.

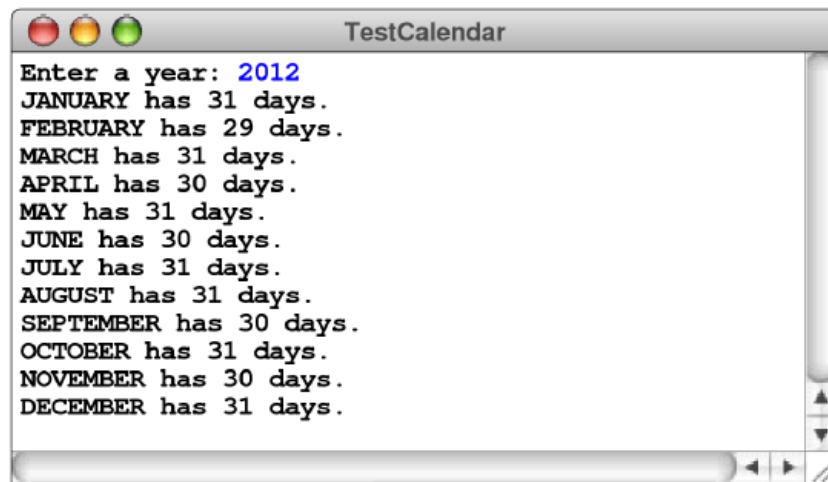
Please note that, the teaching assistant may ask you to explain the meaning of your program, to ensure that the codes are indeed written by yourself. Please also note that we may check whether your program is **too similar** to your fellow students' code using BB.

**Please refer to the BB system for the assignment deadline.** For each day of late submission, you will obtain late penalty in the assignment marks. If you submit more than three days later than the deadline, you will receive zero in this assignment.

**Detailed description on assignment requirement is stated in the last few pages.**

### Question 1:

Using the `direction.h` interface as an example, design and implement a `calendar.h` interface that exports the `Month` type from Chapter 1, along with the functions `daysInMonth` and `isLeapYear`, which also appear in that chapter. Your interface should also export a `monthToString` function that returns the constant name for a value of type `Month`. Test your implementation by writing a main program that asks the user to enter a year and then writes out the number of days in each month of that year, as in the following sample run:



```
TestCalendar
Enter a year: 2012
JANUARY has 31 days.
FEBRUARY has 29 days.
MARCH has 31 days.
APRIL has 30 days.
MAY has 31 days.
JUNE has 30 days.
JULY has 31 days.
AUGUST has 31 days.
SEPTEMBER has 30 days.
OCTOBER has 31 days.
NOVEMBER has 30 days.
DECEMBER has 31 days.
```

## Question 2

The genetic code for all living organisms is carried in its DNA—a molecule with the remarkable capacity to replicate its own structure. The DNA molecule itself consists of a long strand of chemical bases wound together with a similar strand in a double helix. DNA's ability to replicate comes from the fact that its four constituent bases—adenosine, cytosine, guanine, and thymine—combine with each other only in the following ways:

- Cytosine on one strand links only with guanine on the other, and vice versa.
- Adenosine links only with thymine, and vice versa.

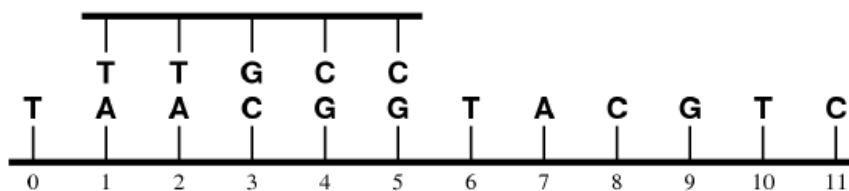
Biologists abbreviate the names of the bases by writing only the initial letter: **A**, **C**, **G**, or **T**.

Inside the cell, a DNA strand acts as a template to which other DNA strands can attach themselves. As an example, suppose that you have the following DNA strand, in which the position of each base has been numbered as it would be in a C++ string:

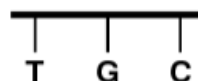


Your mission in this exercise is to determine where a shorter DNA strand can attach itself to the longer one. If, for example, you were trying to find a match for the strand

the rules for DNA dictate that this strand can bind to the longer one only at position 1:



By contrast, the strand



matches at either position 2 or position 7.

Write a function

```
int findDNAMatch(string s1, string s2, int start = 0);
```

that returns the first position at which the DNA strand **s1** can attach to the strand **s2**. As in the **find** method for the **string** class, the optional **start** parameter indicates the index position at which the search should start. If there is no match, **findDNAMatch** should return **-1**.

## Question Requirement

Q1:

You should write four files: **calendar.h**, **calendar.cpp** , **p1.h** and **p1.cpp**.

Of them,

- **calendar.h** should include the declaration of the **Month** type, the functions **daysInMonth** and **isLeapYear**.
- **calendar.cpp** should contains the implementation of the two functions in **calendar.h**.
- **p1.h** should **at least** (You can write other functions to simplify your program) include the declaration of functions **void p1()**.
- **p1.cpp** should include implementations for functions declared in **p1.h**, the test method **void p1()**, which is able to ask the user to input a year and then print the number of days in each month of the year.

The sample output should be:

---

```
Enter a year: 2012
JANUARY has 31 days.
FEBRUARY has 29 days.
MARCH has 31 days.
APRIL has 30 days.
MAY has 31 days.
JUNE has 30 days.
JULY has 31 days.
AUGUST has 31 days.
SEPTEMBER has 30 days.
OCTOBER has 31 days.
NOVEMBER has 30 days.
DECEMBER has 31 days.
```

Q2:

You should write two files: **p2.h** and **p2.cpp**.

Of them,

- **p2.h** should **at least** (You can write other functions to simplify your program) include declaration of the two functions shown below.

```
int findDNAMatch(string s1, string s2, int start = 0);
```

```
void p2();
```

- **p2.cpp** should include implementations for functions declared in p2.h. Of them, p2() is the test method which contains the test program.

The test method void p2(), should be able to ask **user to input** from console to specify the **DNA strand to be attached to and a shorter one**. Then it output **all the positions** after matching.

A sample output should be:

\$Please input the longer DNA strand: TAACGGTACGTC

\$Please input the shorter one: TGC

\$The matching positions should be: 2, 7

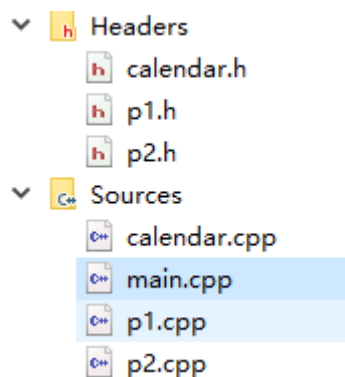
## Assignment Submission

Please put all required files in a single QT project which is stated in the first line of each Question Requirement  
**Write a main.cpp file** which includes a main function to invoke different questions as follows:

```
#include "p1.h"
#include "p2.h"
#include "calendar.h"

int main() {
    p1();
    p2();
    return 0;
}
```

The project layout:



After you test all your questions, zip the whole project in one file named XXX.zip (XXX is your student ID and your name is not required) and then submit it to the BlackBorad (BB) system.

## Marking scheme

For each question:

- **20%** Marks will be given to students who have submitted the program **on time**.
- **20%** Marks will be given to students who wrote the program that meet all the **requirements** of the questions.
- **20%** Marks will be given to students who programs that can be compiled without **errors and warnings**
- **20%** Marks will be given to students whose programs produce the **correct output** if their programs can be compiled.
- **20%** Marks will be given to students who demonstrate good **programming habit and style**.

## **Q&A**

**1. How to check whether the assignment is submitted on time?**

**We directly check it through BB system**

**2. How to check whether the assignment meets the requirement?**

- a. A zip file containing the whole project is submitted**
- b. Zip filename is valid (studentID.zip)**
- c. All the required files stated in Each Question Requirement should be included with correct name.**
- d. The main.cpp with correct content should be included.**
- e. The content of each file should satisfy the requirement. For example, the declaration of p1() should be in p1.h and its implementation should be in p1.cpp.**
- f. No other additional files are included**

**3. How to check whether the assignment can be compiled?**

**Note that we **strongly suggest** you to use the QT IDE since your code tested on other IDE may not work on**



QT. If you want to use other IDE, please test your code again on QT to avoid such errors.

If the QT IDE generates any **errors** when building your project, 20% mark will be missed for each question. If **any warnings** are produced but your code can run, 20% mark will be missed for all questions.

#### 4. How to check the assignment gives correct output?

We will run your code if it can be compiled (warnings are tolerated). You **test code** should be correct and **output** is also correct. User-friendly test code and output is preferred.

#### 5. How to check the programming style?

- **Comments.**

You should include three kinds of comments in your code:

a. File comments in the beginning of each code file, for example,

```
/*
 * File: AddIntegerList.cpp
 * -----
 * This program adds a list of integers. The end of the
 * input is indicated by entering a sentinel value, which
 * is defined by setting the value of the constant SENTINEL.
 */

#include <iostream>
using namespace std;
..
```

b. Function comments just before each function implementation. For example,

```
/*  
 * Function: raiseToPower  
 * Usage: int p = raiseToPower(n, k);  
 * -----  
 * Returns the integer n raised to the kth power.  
 */  
  
int raiseToPower(int n, int k) {  
    int result = 1;  
    for (int i = 0; i < k; i++) {  
        result *= n;  
    }  
    return result;  
}
```

c. Statement comments on the right side of some statements **if necessary**. You do not have to write comments for each statement.

- Code layout.

**Beautiful !!!!**

The following points should be pay attention to:

- a. The indentation used
- b. {} used for function, for structure, while structure, switch structure.
- c. meaningful variable name.

We strongly suggest you to adhere to the coding style used in our textbook, which will be followed by us to check your code.