Homework 2: Overloading in C++

- C++ Environment for grading is **Dev-C++ 5.11** (C++ Compiler: **TDM-GCC 4.9.2**)
- C++ time library is not allowed in this homework.
- Please ensure that your code files can be complied correctly before submitting on YZU Portal. Otherwise, there will be 0 points for that task.
- Submissions that are more than 90% similar will be reduced 5% increasingly based on the submission time.
- The **main.cpp** file includes test cases will be provided soon for self-testing, there is some hidden test cases will be used for grading.
- Penalty for late parts: -10% of value for each day late.

Task 1 (40 points)**:** Create class **Time** which contains 3 private data members as integers: **second**, **minute**, **hour** following these requirements:

1. (5 points) Create these constructors with/without arguments:

```
Time(); // hour = minute = second = 0
Time(int hour, int minute, int second);
```

Note: Be sure that hour in range 0 to 23, minute and second in range 0 to 59. Remember to consider all cases when hour/minute/second increases/decreases over the range, then add/subtract 1 unit to higher time metrics.

```
Time t(23,59,59);
++t;   // add 1 second then return 0:0:0
```

2. (5 points) Create public **getters** and **setters** to assign values of **Time** class.

```
int getSecond();
int getMinute();
int getHour();
void setSecond(int second);
void setMinute(int minute);
void setHour(int hour);
```

3. (5 points) Create **add** function to add or subtract amount of time unit from current object:

```
void add(int amount, string unit);
```

Example:

```
Time t;
t.add(5, "second");     // add 5 seconds
t.add(2, "minute");     // add 2 minutes
t.add(1, "hour");       // add 1 hour
```

```
t.add(-12, "second");   // subtract 12 seconds
t.add(-3, "minute");    // subtract 3 minutes
t.add(-2, "hour");      // subtract 2 hours
```

4. (5 points) Create **duration** function which calculates the time difference of two **Time** objects in units of seconds.

```
Time t1(2,23,51);
Time t2(2,24,3);
t1.duration(t2);    // 12 seconds
t2.duration(t1);    // 12 seconds
```

5. (5 points) Create relational operator overloading to compare two **Time** objects: `<, >, ==, <=, >=, !=`

   Example: `t1 > t2    // return boolean true/false`

6. (5 points) Create ++ and -- operators overloading by adding or subtracting number of seconds to current **Time** object.
   Example:

```
Time t(2,23,51);
++t;     // add 1 second to t
--t;     // subtract 1 second to t
t += 5;  // add 5 seconds to t
t -= 8   // subtract 8 seconds to t
```

7. (10 points) Create – operator between two **Time** objects, the result returns time difference of two-Time objects in units of seconds.

```
Time t1(2,23,51);
Time t2(2,24,3);

int a = t2 - t1;    // a = 12  (seconds)
int b = t1 - t2     // b = -12 (seconds)
```

Task 2 (60 points)**:** Create class **DateTime** which contains 4 private data members: **date**, **month**, **year** as integers and **time** as **Time** object from Task 1.

1. (5 points) Create constructors with/ without argument and with arguments.

```
DateTime(); // set date/month/year to 1/1/1900 and time object to 0:0:0
DateTime(int date, int month, int year); // set time object to 0:0:0
DateTime(int date, int month, int year, Time time);
DateTime(int date, int month, int year, int hour, int min, int second);
```

   Note: Ensure that year is positive number, month is in range 1 to 12, date is in range 1 to 31 which depends on current month. Remember to consider all cases when date/month/year/hour/minute/second increases over the range, then add 1 unit to higher date or time metrics, and **leap year** (*check example in Task 2-part j*).

2. (5 points) Create **getters** and **setters** to assign values of **DateTime** class.

```
int getDate();
 int getMonth();
 int getYear();
 int getSecond();
 int getMinute();
 int getHour();
 void setDate(int date);
 void setMonth(int month);
 void setYear(int year);
 void setSecond(int second);
 void setMinute(int minute);
 void setHour(int hour);
```

3. (5 points) Create **add** function to add or subtract amount of date or time unit from current object:

```
void add(int amount, string unit);
```

Example:

```
Time t;
t.add(1, "year");       // add 1 year
t.add(-2, "month");     // subtract 2 months
t.add(3, "date");       // add 1 dates
t.add(2, "week");       // add 2 weeks
t.add(-12, "second");   // subtract 12 seconds
t.add(-3, "minute");    // subtract 3 minutes
t.add(-2, "hour");      // subtract 2 hours
```

4. (5 points) Create **dayOfMonth** function which returns the number of day in the given month, for Example:

```
DateTime d1(1,12,2023)
d1.dayOfMonth();     // return 31
```

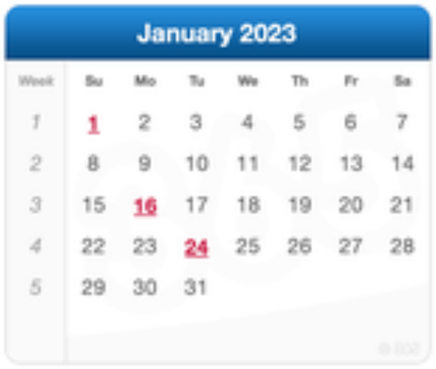
5. (5 points) Create **dayOfYear** function which returns the number of day that year already passed, for Example:

```
DateTime d1(1,1,2023);
d1.dayOfYear();     // return 1
DateTime d2(2,2,2023)l
d2.dayOfYear();     // return 33.
```

6. (5 points) Create **dayOfWeek** function which returns the day in week.
   Example:

```
DateTime d1(3,5,2023);
d1.dayOfWeek();     // return Wednesday
```

7. (5 points) Create **weekOfYear** function which returns the number of current week in that year, noticed that the begin of a week start from Sunday (if you till confuse, please check the picture below).



January 2023

Week	Su	Mo	Tu	We	Th	Fr	Sa
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21
4	22	23	24	25	26	27	28
5	29	30	31				

Example:

```
DateTime d1(2,2,2023);
d1.weekOfYear();     // return week 5
```

8. (5 points) Create **quaterOfYear** function which returns the current quarter of that year.

   Example:

```
DateTime d1(31,12,2022)
d1.quaterOfYear();     // return quarter 4
```

9. (5 points) Create **relational operator overloading** to compare two **DateTime** objects: `<, >, ==, <=, >=, !=`. Students can reuse operator overloading from Time object in Task 1

10. (5 points) Create ++ and -- operators overloading by adding or subtracting number of days to current **DateTime** object.

    Example:

```
DateTime dt1(28,2,2023);
++dt1;     // add 1 day to dt1, then return 1/3/2023
DateTime dt2(1,1,2023);
--dt2;     // subtract 1 day to dt2, then return 31/12/2022
```

11. (5 points) Create **duration** function which calculate the time difference of two **DateTime** objects in units of days.

```
DateTime dt1(3,5,2023,12,0,0);
DateTime dt2(5,5,2023);
dt1.duration(dt2);  // 1.5 days
dt2.duration(dt1);  // 1.5 days
```

12. (5 points) Create – operator between two **DateTime** objects, the result returns time difference of two **DateTime** objects in units of days.

```
DateTime dt1(3,5,2023,12,0,0);
DateTime dt2(5,5,2023);
float a = dt2 - dt1;   // a = 1.5  (days)
float b = dt1 - dt2;   // b = -1.5 (days)
```