COMP 2012H Honors Object-Oriented Programming and Data Structures

Assignment 1 Calendar

Honor Code

We value academic integrity very highly. Please read the Honor Code section on our course webpage to make sure you understand what is considered as plagiarism and what the penalties are. The following are some of the highlights:

- Do NOT try your "luck" - we use sophisticated plagiarism detection software to find cheaters. We also review codes for potential cases manually.
- The penalty (for **BOTH** the copier and the copiee) is not just getting a zero in your assignment. Please read the Honor Code thoroughly.
- Serious offenders will fail the course immediately, and there will be additional disciplinary actions from the department and university, upto and including expulsion.

End of Honor Code

Objectives & Intended Learning Outcomes

The objective of this assignment is to provide you with practice on C++ basic variables and control flows. Upon completion of this assignment, you should be able to:

1. Use basic variables in C++, like int, char, bool, etc.
2. Use if, if-else, switch statements to control program flow
3. Use relational operators and logical operators to control program flow.
4. Use for, while, and do-while to write loops, as well as use break and continue to exit loops.

End of Objectives & Intended Learning Outcomes

Source: https://siliconvalleyyouthbridge.org/calendar/

Introduction

Page maintained by

CHEN, Jierun
Email:
jierunchen@ust.hk
Last Modified:
09/04/2022 13:30:07

Homepage

Course Homepage

The Gregorian calendar is the de facto international standard and is used almost everywhere in the world for civil purposes. The widely used solar aspect is a cycle of leap days in a 400-year cycle designed to keep the duration of the year aligned with the solar year.

The Gregorian calendar has also been implemented in Linux/Unix command: `cal` and `ncal`. If you are using Mac or Linux computers, you may try these commands in your terminal:

```
> cal -m 9 2022
   September 2022
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30

> ncal -m 9 2022
    September 2022
Mo     5 12 19 26
Tu     6 13 20 27
We     7 14 21 28
Th  1  8 15 22 29
Fr  2  9 16 23 30
Sa  3 10 17 24
Su  4 11 18 25

> cal 2023
                            2023
      January               February              March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7            1  2  3  4            1  2  3  4
 8  9 10 11 12 13 14   5  6  7  8  9 10 11   5  6  7  8  9 10 11
15 16 17 18 19 20 21  12 13 14 15 16 17 18  12 13 14 15 16 17 18
22 23 24 25 26 27 28  19 20 21 22 23 24 25  19 20 21 22 23 24 25
29 30 31              26 27 28              26 27 28 29 30 31


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

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

      October               November              December
```

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

In this assignment, you are going to implement such a tool on your own to display and format calendars of different styles! More details can be found below.

End of Introduction

Description

Please read the FAQ section for some common clarifications. You should check that a day before the deadline to make sure you don't miss any clarification, even if you have already submitted your work by then.

Gregorian calendar

In this assignment, we focus on Gregorian calendar only, you may need to read the Wikipedia page https://en.wikipedia.org/wiki/Gregorian_calendar on what it is, and most importantly, **how many days are there in each month, especially February.** The way of calculating leap year **might be different** from what you have learnt before.

Program Overview

After the program is started, it will ask user to input a **year** (in range 1800-9999), and then input a **month** (in range 1-12). If the user input is not valid, the program will continue asking the user to input until a valid one is received.

Next, the program will ask the user whether he/she would like to use Monday or Sunday as the first day of a week. If user chooses Monday, then the calendar displayed later will start each week on Monday, and vice versa. If the user input is not valid, the program will continue asking the user to input until a valid one is received.

Then, the program will ask the user if they want the calendar to print horizontally or vertically. If user chooses horizontally, then the calendar will look like the one using `cal` command on Linux, otherwise it will look like the one using `ncal` command. Again, if the user input is not valid, the program will continue asking the user to input until a valid one is received.

Note that in bonus tasks, the user may input `-1` or `-3` for month. In this situation, the next two options (start week on Monday/Sunday, and print horizontally/vertically) will be disabled. The program will directly print calendar after user enters the month.

How to calculate which day it is for a certain date?

The only thing you have is that **the January 1st of year 1800 is a Wednesday**. All other you need is in the Wikipedia link given in Gregorian calendar part above. **Actually you only need to figure out how many days are there in each month.**

You are expected to **calculate** which day it is for a certain date when you display the calendar later.

The calendar rules used before 1800 are different to the rules we use today, due to some historic reasons. So in this PA, you only need to work on calendars **after year 1800**.

Additional Remarks

In this assignment, you can feel free to use anything you know, and there is no need to restrict yourself to only use what we have taught so far. For example, if you think using a function will make your codes tidy, feel free to do it. However, **you can only use header file `#include <iostream>`** .

If you would like to try the bonus tasks, it is very likely that you will find functions useful. Otherwise, your codes might look quite messy.

End of Description

Tasks

This section describes all you will need to implement. You may need to check Program Overview section to have an overview of this program before reading this part.

Task 1 - Receive and validate user input

As described in Program Overview, ask the user to input (1) year, (2) month, (3) start each week on Monday or Sunday, and (4) print horizontally or vertically.

If the user input is not valid, the program will keep asking the user to input until a valid one is received. You don't need to consider the cases when user input incorrect data types. For example, if we want an integer, you can assume the user will always input an integer, not a character, etc.

For students who do bonus part: You should skip (3) and (4) if user input month -1 or -3, as described in Bonus Tasks.

Example: (user input is displayed in red)

```
Enter year (in range 1800-9999): 123
Enter year (in range 1800-9999): 12345
Enter year (in range 1800-9999): 1800
Enter month (in range 1-12): -100
Enter month (in range 1-12): 100
Enter month (in range 1-12): 1
Enter the first day of week ('s' for Sunday, 'm' for Monday): k
Enter the first day of week ('s' for Sunday, 'm' for Monday): o
Enter the first day of week ('s' for Sunday, 'm' for Monday): s
Print horizontally (h) or vertically (v): c
Print horizontally (h) or vertically (v): o
Print horizontally (h) or vertically (v): m
Print horizontally (h) or vertically (v): p
Print horizontally (h) or vertically (v): h
[Start printing the calendar, remaining parts are ignored. See examples in
other tasks]
```

Here are some code snippets that you can copy and paste to your program, if you find it convenient.

```
cout << "Enter year (in range 1800-9999): ";
cout << "Enter month (in range 1-12): ";
cout << "Enter the first day of week ('s' for Sunday, 'm' for Monday): ";
cout << "Print horizontally (h) or vertically (v): ";
```

Task 2 - Horizontal calendars with week starts on Sunday

When user chooses to start each week on Sunday and print the calendar horizontally, display the calendar in such style.

- First a title should be displayed, with format "[Month Name] [Year]", for example, "January 1800". To make it easier, please make the title left-aligned.
- Then, display the names of weekdays, from "Su" to "Sa", words should be separated by one space.
- Finally, format the calendar like the example below. Dates should be separated by one space, and for dates with only one digit, the digit should be right-aligned.
- After printing the calendar, the program finishes.
- It's ok to have extra spaces **at the end of** each line. See details in Grading section

Note: This should be the same as Linux command `cal -m [month] [year]`, except that you don't need to align the title or ouput additional blank line at the end.

Example: (user input is displayed in red)

```
Enter year (in range 1800-9999): 1800
Enter month (in range 1-12): 1
Enter the first day of week ('s' for Sunday, 'm' for Monday): s
Print horizontally (h) or vertically (v): h
January 1800
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

Task 3 - Horizontal calendars with week starts on Monday

When user chooses to start each week on Monday and print the calendar horizontally, display the calendar in such style.

- First a title should be displayed, with format "[Month Name] [Year]", for example, "January 1800". To make it easier, please make the title left-aligned.
- Then, display the names of weekdays, from "Mo" to "Su", words should be separated by one space.
- Finally, format the calendar like the example below. Dates should be separated by one space, and for dates with only one digit, the digit should be right-aligned.
- After printing the calendar, the program finishes.
- It's ok to have extra spaces **at the end of** each line. See details in Grading section

Example: (user input is displayed in red)

```
Enter year (in range 1800-9999): 1800
Enter month (in range 1-12): 1
Enter the first day of week ('s' for Sunday, 'm' for Monday): m
Print horizontally (h) or vertically (v): h
January 1800
Mo Tu We Th Fr Sa Su
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

Task 4 - Vertical calendars with week starts on Sunday

When user chooses to start each week on Sunday and print the calendar vertically, display the calendar in such style.

- First a title should be displayed, with format "[Month Name] [Year]", for example, "January 1800". To make it easier, please make the title left-aligned.
- Then, display the names of weekdays, from "Su" to "Sa", **as the first column on the left.** Those names and dates you will print later (remaining columns) should be separated

by one space.
- Finally, format the calendar **vertically,** like the example below. Dates should be separated by one space, and for dates with only one digit, the digit should be right-aligned.
- After printing the calendar, the program finishes.
- It's ok to have extra spaces **at the end of** each line. See details in Grading section

Example: (user input is displayed in red)

```
Enter year (in range 1800-9999): 1800
Enter month (in range 1-12): 1
Enter the first day of week ('s' for Sunday, 'm' for Monday): s
Print horizontally (h) or vertically (v): v
January 1800
Su     5 12 19 26
Mo     6 13 20 27
Tu     7 14 21 28
We  1  8 15 22 29
Th  2  9 16 23 30
Fr  3 10 17 24 31
Sa  4 11 18 25
```

Task 5 - Vertical calendars with week starts on Monday

When user chooses to start each week on Monday and print the calendar vertically, display the calendar in such style.

- First a title should be displayed, with format "[Month Name] [Year]", for example, "January 1800". To make it easier, please make the title left-aligned.
- Then, display the names of weekdays, from "Mo" to "Su", **as the first column on the left.** Those names and dates you will print later (remaining columns) should be separated by one space.
- Finally, format the calendar **vertically,** like the example below. Dates should be separated by one space, and for dates with only one digit, the digit should be right-aligned.
- After printing the calendar, the program finishes.
- It's ok to have extra spaces **at the end of** each line. See details in Grading section

Note: This should be the same as Linux command `ncal -M [month] [year]`, except that you don't need to align the title or ouput additional blank line at the end.

Example: (user input is displayed in red)

```
Enter year (in range 1800-9999): 1800
Enter month (in range 1-12): 1
Enter the first day of week ('s' for Sunday, 'm' for Monday): m
Print horizontally (h) or vertically (v): v
January 1800
Mo     6 13 20 27
Tu     7 14 21 28
We  1  8 15 22 29
Th  2  9 16 23 30
Fr  3 10 17 24 31
Sa  4 11 18 25
Su  5 12 19 26
```

Bonus Task 1 - Print calendar of a whole year

This is a bonus task, please see grading details in Grading section

When user chooses to print the calendar of a whole year, indicating by entering -1 as month, we will start each week on Sunday and print the calendar horizontally.

- First a title should be displayed, with format "Year [Year no.]", for example, "Year 1800". To make it easier, please make the title left-aligned.
- Next, **for each month:**
  - A sub-title should be displayed, with format "[Month Name]", for example, "January". To make it easier, please make it left-aligned.
  - Display the names of weekdays, from "Su" to "Sa", words should be separated by one space.
  - Finally, format the calendar like the example below. Dates should be separated by one space, and for dates with only one digit, the digit should be right-aligned.
- You should print three months horizontally. Two adjacent months in the same row should be separated by two spaces.
- If any of the horizontal three months has 6 rows enumerating the date, the remaining horizontal months should also have 6 rows for alignment.
- There is one blank line between two set of horizontal three months.
- After printing the calendar, the program finishes.
- It's ok to have extra spaces **at the end of** each line. See details in Grading section

Note: This should be the same as Linux command `cal [year]`, except that you don't need to align the title and sub-titles or ouput additional blank line at the end.

Example: (user input is displayed in red)

```
Enter year (in range 1800-9999): 1800
Enter month (in range 1-12): -1
Year 1800
January               February              March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4                     1                     1
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   2  3  4  5  6  7  8
12 13 14 15 16 17 18   9 10 11 12 13 14 15   9 10 11 12 13 14 15
19 20 21 22 23 24 25  16 17 18 19 20 21 22  16 17 18 19 20 21 22
26 27 28 29 30 31     23 24 25 26 27 28     23 24 25 26 27 28 29
                                            30 31

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

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

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

Bonus Task 2 - Print calendar of adjacent 3 months

This is a bonus task, please see grading details in Grading section

When user chooses to print the calendar of adjacent 3 months (last month, current month, and next month), indicating by first enter a valid year, and then enter -3 as month. Then, the program will ask which month do the user want to print as current month (note that we only have this prompt in this task). After that, we will start each week on Sunday and print the calendar horizontally.

Note: Since we only process year 1800-9999, so if user want to print adjacent 3 months, with Jan 1800 or Dec 9999 as current month, we will consider the case invalid, and your program should ask user to input again.

- You should print the three months in the same row. Two adjacent months in the same row should be separated by two spaces. For each month,
  - A sub-title should be displayed, with format "[Month Name] [Year]", for example, "August 2022". To make it easier, please make it left-aligned.
  - Display the names of weekdays, from "Su" to "Sa", words should be separated by one space.
  - Finally, format the calendar like the example below. Dates should be separated by one space, and for dates with only one digit, the digit should be right-aligned.
- After printing the calendar, the program finishes.
- It's ok to have extra spaces **at the end of** each line. See details in Grading section

Note: This should be the same as Linux command `cal -3 -m [center month] [year]`, except that you don't need to align the sub-titles or ouput additional blank line at the end.

Example: (user input is displayed in red)

```
Enter year (in range 1800-9999): 1800
Enter month (in range 1-12): -3
Enter the month in the center (in range 1-12): 2
January 1800          February 1800         March 1800
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4                     1                     1
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   2  3  4  5  6  7  8
12 13 14 15 16 17 18   9 10 11 12 13 14 15   9 10 11 12 13 14 15
19 20 21 22 23 24 25  16 17 18 19 20 21 22  16 17 18 19 20 21 22
26 27 28 29 30 31     23 24 25 26 27 28     23 24 25 26 27 28 29
                                            30 31
```

End of Tasks

Resources & Sample I/O

- Skeleton code: **No skeleton for this PA, you are expected to start from scratch.**
- Demo programs: Windows / MacOS Intel / Linux (compiled on CS Lab 2 Machine)
- Revealed test cases: Download here. They are the same as examples in Tasks section. Please note that test cases on ZINC before deadline are the same as those examples.
  Additional note: You may notice that the format of test cases downloaded is not the same as the format you see in your terminal. This is because when we test your program, we will let your program read input from a file and output to another file. Please refer to this section about how you can test your program by yourself.
  - Sample 1: horizontal, start weeks on Sunday
  - Sample 2: horizontal, start weeks on Monday
  - Sample 3: vertical, start weeks on Sunday
  - Sample 4: vertical, start weeks on Sunday
  - Sample 5: Bonus 1: print whole year
  - Sample 6: Bonus 2: print adjacent 3 months

End of Resources & Sample I/O

Submission & Grading

Deadline: Sat, 17/9/22 HKT 23:59.
Best 10 out of 11 PAs will be counted towards your course grade, each with 6.5% at most via Automated Grading on the ZINC Online Submission System. Compress the single source code file `main.cpp` by itself as `PA1.zip` for submission to ZINC.

Grading Scheme

The grading process will be performed entirely on ZINC Online Submission System. We will input some contents to your program, and **simply compare its output with expected output**. To avoid being too demanding, ZINC will consider the results are identical as long as they "look exactly the same". To be more specific, **trailing spaces at the end of each line and a newline character at the end of the file will be ignored.**

For example, if two lines are `"1 2 3 4"` and `"1 2 3 4  "`, they will be considered identical. However, if two lines are `"1 2"` and `"1  2"`, they are not the same, since we only ignore spaces **at the end of each line/the file**.

Before the deadline, the test cases on ZINC will only contain a few cases. If you passed those tests, it only means your program can successfully run on ZINC and can pass those test cases. The result is by no means complete and the score is irrelavant to your actual score. In the actual grading stage after the deadline, we will use a **totally different set of test cases**, which is expected to be more complete and more strict, to test the correctness of your program.

Please ensure that you submit to ZINC well before the deadline as **all late submissions will be automatically rejected**.

Note: When testing Task 1-5, we guarantee that there won't be input where month is -1 or -3. So even if you complete Bonus part, your grade in Task 1-5 will not be affected.

How to test my program

You may compile your code using this command: (the `-Wall` flag is optional, it will show all warnings to you)

```
g++ -o main main.cpp -std=c++11 -Wall
```

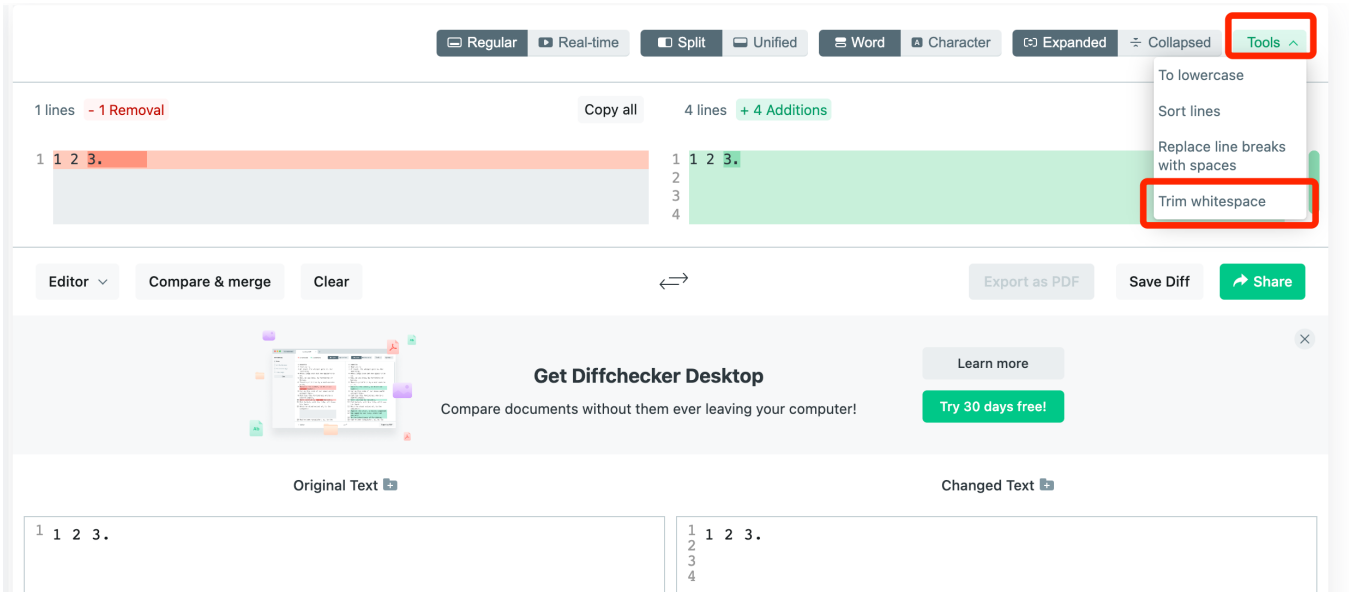
Next, please place `main, input.txt, output.txt` (i.e., the sample input/output that we provided) in the same folder, and let your program read input from file `input.txt` and output result to file `my_output.txt` by using this command: (you can change the filename, e.g. input3.txt and output3.txt, etc.)

```
./main < input.txt > my_output.txt
```

If you are using Windows PowerShell, you may use the following command instead:

```
Get-Content input.txt | ./main > my_output.txt
```

Then you may check if your output in `my_output.txt` is the same as the expected output `output.txt` by using this online diff checker tool



By default it will not ignore trailing spaces and blank lines, you need to click "Tools" at the top-right, then "Trim whitespace", as the image shows.

Alternative: If you are using a Linux machine (MacOS is not ok) or using CS Lab 2 machines, you can directly use this command:

```
diff -Z -B output.txt my_output.txt
```

This command will automatically ignore trailing spaces and blank lines. If you see nothing after executing this command, then the contents in these two files is considered the same.

End of Submission & Grading

Frequently Asked Questions

Q: My code doesn't work, there is an error/bug, here is the code, can you help me fix it?
A: As the assignment is a major course assessment, to be fair, you are supposed to work on it on your own and we should not finish the tasks for you. We are happy to help with explanations and advice, but we shall not directly debug the code for you.

Q: The demo program enters an infinite loop when given unexpected input (e.g. inputting a character when expecting an integer). Is this a bug?
A: This is just the behavior of `cin >> variable;` when given input that is not type-matched. You don't need to worry about such for PA1, you can assume the user always input the correct data type. For example, if we want an `int`, the user will always input an `int`, not other weird things. However, the user input can still be invalid, for example, you want an integer between 1800-9999, and the user may input 1700, or 12345, etc.

Q: What are the restrictions regarding modifying the header files, writing our own helper functions, including extra header files, etc.?
A: The only hard restriction is that you can only submit `main.cpp` to ZINC, and include only one header file `#include <iostream>`. Anything else are not strictly prohibited, as long as your code can run on ZINC.

End of FAQ

Acknowledgement - The assignment 1 is drafted by LIU Jianmeng.

LIU, Jianmeng