Algorithms and Data Structures
Jacobs University Bremen
Dr. Kinga Lipskoch

Course: CH-231-A
February 10th, 2020

Homework 2

- Submit one ZIP file per homework sheet which contains one PDF file (including pictures, computations, formulas, explanations, etc.) and your source code file(s) with one makefile and without adding executable, object or temporary files.
- The implementations of algorithms has to be done using C, C++, Python or Java.
- The TAs are grading solutions to the problems according to the following criteria: https://grader.eecs.jacobs-university.de/courses/ch_231_a/2020_1/Grading_Criteria_ADS.pdf

Problem 2.1 *Using lists* (1 point)

Language: C++

Write a program which does the following using two `list` objects:

1. Create two lists (`A` and `B`).
2. Read integers from the keyboard until the entered integer is negative or zero.
3. Insert the positive integers into list `A` by adding to the end.
4. Insert the same positive integers into list `B` by adding to the beginning.
5. Print list `A` (separated by spaces) on the standard output and print list `B` (separated by spaces) into a file called "`listB.txt`".
6. Print an empty line on the standard output.
7. Move the first element of the lists to the end (for both lists).
8. Print list `A`, print list `B` on the standard output (both separated by comma) using an iterator. Make sure that you do not print a comma after the last element.
9. Print an empty line on the standard output.
10. Merge list `B` into list `A`.
11. Print the result of the merging as a sorted list on the standard output (separated by spaces).

You can assume that the input will be valid.

Problem 2.2 *Using deques* (1 point)

Language: C++

Write a program which does the following using a `deque` object:

1. Create a deque `A` able to store float values.
2. Read floats from the keyboard until the entered float value is `0`.
3. Insert the positive elements at the end of `A` and the negative elements at the beginning of `A`.
4. Print the elements of `A` on the standard output separated by spaces.
5. Print an empty line on the standard output.
6. Add the value `0` into the middle of the deque (between the last negative and before the first positive element).
7. Print the elements of `A` on the standard output separated by semicolons. Make sure that you do not print a semicolon after the last element.

You can assume that the input will be valid.

Problem 2.3 *Another deque* (1 point)

Language: C++

A company that provides a special service for wind surfers installs wind gauges at popular windsurfing locations. Each gauge reports the current wind speed to a central computer every 5 minutes. When a user connects to the service she or he is able to retrieve recent high, low, and average wind speeds for her or his selected location. The central computer creates a `WindGauge` object for each gauge. The `WindGauge` class interface looks as follows:

```
class WindGauge {
  public:
    WindGauge(int period = 12);
    void currentWindSpeed(int speed);
    int highest() const;
    int lowest() const;
    int average() const;
  private:
    // add properties and/or method(s) here
};
```

The constructor argument specifies how much history is retained by the object (default 12 periods which is 1 hour). When `currentWindSpeed()` is called, the current wind speed is added to the history. If the history is then longer than the specified period, the oldest wind speed is discarded. The other three functions return the highest, lowest, and average wind speeds reported during the history period.

Implement the `WindGauge` class as specified above. Add properties to the class and write a dump function that prints out the lowest, highest, and average wind speed for a `WindGauge`'s data. Write a test program that does the following:

1. Create a `WindGauge` object.
2. Add five wind speeds: 15, 16, 12, 15, and 15, and then dump the gauge.
3. Add ten more measurements: 16, 17, 16, 16, 20, 17, 16, 15, 16, and 20 (bringing the total to over 12) and dump the numbers again.

Separate your code into three files: `WindGauge.h` (class definition), `WindGauge.cpp` (implementation of methods) and `testWindGauge.cpp` (test program).

Problem 2.4 *Reversing a vector* (1 point)

Language: C++

Write a program which fills a vector with the integer values from 1 to 30. Then add the value 5 at the end of the vector. Reverse the vector using the `reverse()` function from the `algorithm` library and print the content of the vector on the standard output using an iterator. Then replace all occurrences of the value 5 by the value 129 using the `replace()` function from the `algorithm` library and print the modified vector again on the standard output.

Problem 2.5 *Lotto* (2 points)

Language: C++

In the lottery 6 out of 49 numbers are randomly drawn. Draw six different numbers using the formula `rand() % 49 + 1`. Then add the drawn number to a container that stores all drawn numbers (but make sure that your container will not contain duplicates).

After you have drawn all numbers, print them on the standard output in ascending order. Use a suitable STL container that supports the needed operations and corresponding iterators for all actions. Make sure that you initialize the random number generator with the local time of your system at the beginning of your program.

Bonus Problem 2.6 *Another set* (1 point)

Language: C++

A building security system has door locks that are opened by typing a four-digit access code into a keypad. The access code is validated by querying an `Access` object with the following interface:

```
class Access {
  public:
    void activate(unsigned int code);
    void deactivate(unsigned int code);
    bool isactive(unsigned int code) const;
  private:
    // add properties and/or method(s) if necessary
};
```

Each employee is given a different access code, which is activated using the `activate()` method. When an employee leaves the company, his or her access code is deactivated using the `deactivate()` method.
Implement the `Access` class specified above. Write a test program that does the following:

1. Create an `Access` object.
2. Activate the codes 1234, 9999, and 9876.
3. Ask the user for an access code. Read the code from the keyboard.
4. Tell the user whether the door opened successfully.
5. Repeat the last two steps until the door opens.
6. Deactivate the code that worked. Also, deactivate code 9999 and activate code 1111.
7. Repeat steps 3 and 4 until the door opens.

Separate your code into three files: `Access.h` (class definition), `Access.cpp` (implementation of methods) and `testAccessSet.cpp` (test program).
You can assume that the input will be valid.

Problem 2.7 *Using maps* (2 points)

Language: C++
Write a program which creates a collection of names and birthday dates. Your program should read the data from a file called "`data.txt`" which contains a name followed by a corresponding birthday date on different lines repeated for many persons. A name consists of first name and last name separated by space. Your program should read the content of the file and use a `map` to store names and birthday dates.
Then "simulate" querying your collection (i.e., the map) by asking for a name from the keyboard and printing on the standard output the corresponding birthday date. If the name is not in your container then print "`Name not found!`" on the screen.
You can assume that the input will be valid and the content of the file will be valid if existing.

Bonus Problem 2.8 *Another map* (2 points)

Language: C++
The customer that uses the security system described in the previous exercise now wants to associate an access level with each access code. Users with high access levels can open doors to more security-sensitive parts of the building than users with lower access levels. Begin with your solution to **Problem 2.6**. Modify the `Access` class from the previous exercise so that an integer access level is associated with each code. The new interface should be as follows:

```
class Access {
  public:
    void activate(unsigned int code, unsigned int level);
    void deactivate(unsigned int code);
    bool isactive(unsigned int code, unsigned int level) const;
  private:
    // add properties and/or method(s) if necessary
};
```

The `isactive()` method returns `true` if the specified access code has an access level greater than or equal to the specified access level. Of course, it should return `false` if the access code is not active at all. Modify the main program to do the following:

1. Create an `Access` object.
2. Activate code 1234 with access level 1, code 9999 with access level 5, and code 9876 with access level 9.
3. Ask the user for an access code. Read the code from the keyboard.
4. Assuming a door that requires access level 5 for entry, tell the user whether the door opened successfully.
5. Repeat the last two steps until the door opens.
6. Deactivate the code that worked. Change the access level of code 9999 to 8. Activate code 1111 with access level 7.
7. Ask the user for an access code. Read the code from the keyboard.
8. Assuming a door that requires access level 7 for entry, tell the user whether the door opened successfully.
9. Repeat the last two steps until the door opens.

Separate your code into three files: `Access.h` (class definition), `Access.cpp` (implementation of methods) and `testAccessMap.cpp` (test program).
You can assume that the input will be valid.

How to submit your solutions

You can submit your solutions via *Grader* at **`https://grader.eecs.jacobs-university.de`** as a generated PDF file and/or source code files.
If there are problems with *Grader* (but only then), you can submit the file by sending mail to `k.lipskoch@jacobs-university.de` **with a subject line that starts with CH08-320201**.

Please note, that after the deadline it will not be possible to submit solutions. It is useless to send solutions by mail, because they will not be graded.

This homework is due by Monday, February 17th, 23:00.