CS 201, Spring 2020

Homework 1

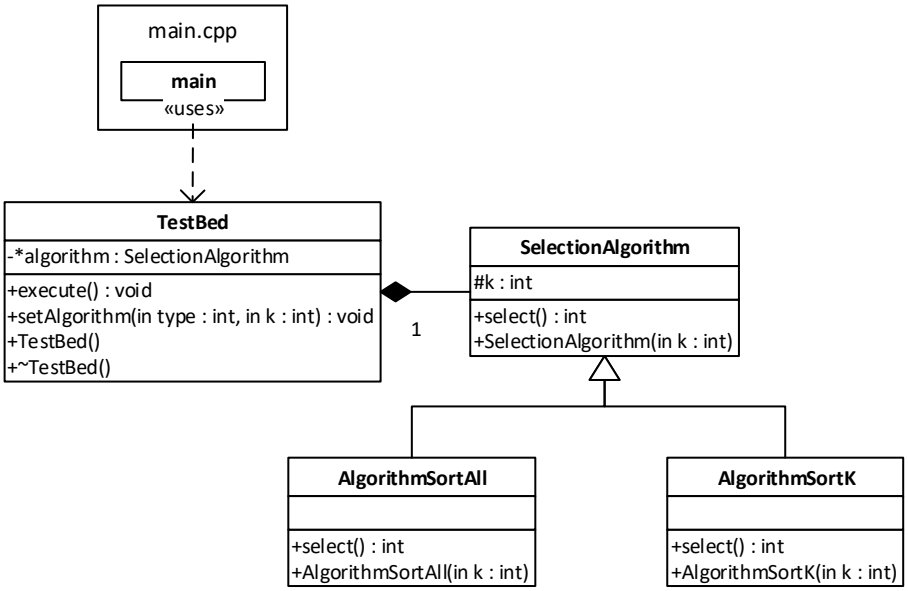
DUE: March 6, Friday @ 23:59.

Please check the submission rules at the end of the document. Points will be deducted in case of a violation of these rules!

Description: In this assignment you will write a C++ program that **finds the *kth* largest number among a set of *N* numbers**. It will implement the solution using **two different algorithms** and measure the time elapsed during the execution of these algorithms. The design of the program will follow the **Strategy Pattern** [1] to be able to switch algorithms at runtime. As input, the program will take the **type of algorithm** to be applied (1 or 2), ***k*** (a number less than or equal to *N*). Then it will take ***N*** followed by a list of ***N* numbers**. All these input values will be obtained from an input file. As output, it will print out the ***kth* largest number** and the **total elapsed time** for the completion of the algorithm. The details are provided in the following sections.

1) The Design

The program will be composed of **four classes and a main function** as depicted below:



The *main* function will create an object of type *TestBed* and it will first consume **two inputs** (i.e., via the `cin` command as explained later in Section 5) to obtain; *i)* the type of the algorithm (can be either 1 or 2), and *ii) k*. It will set up the test bed accordingly, using the *setAlgorithm* method. At the end of the main function, the *TestBed* object should be deleted.

TestBed class has a member **pointer** variable, **algorithm* of type *SelectionAlgorithm*. The *setAlgorithm* method is supposed to initialize this pointer by creating an object of the class *AlgorithmSortAll* or *AlgorithmSortK* depending on the chosen algorithm type (1 or 2, respectively). The *execute()* method calls the *select()* method (i.e., the call is made as "*algorithm->select()"*, while the actual method being called depends on the active object instance). This method also measures the time elapsed during the execution of the *select()* method and prints the result. The *destructor* of this class should free the resources by deleting the object pointed by **algorithm*.

SelectionAlgorithm is an abstract class that is not directly used. It has a protected member variable *k*, which is initialized by the constructor. The *select()* method is actually implemented by the subclasses of the *SelectionAlgorithm* class (i.e., it should be declared as *virtual*). The method takes *N*, and then a total of *N* numbers via the *cin* command (these values are not provided as arguments) and prints out the *kth* largest element. The constructors of the subclasses of the *SelectionAlgorithm* class directly call the constructor of their superclass.

You can make minor modifications to these design guidelines if you like, e.g., having additional methods, member variables, arguments, etc. However, the overall design should conform to the provided guidelines.

2) The Algorithms

The program will implement the following two algorithms in the corresponding subclasses of the *SelectionAlgorithm* class:

Algorithm 1 (AlgorithmSortAll):

- Store all the numbers in an array;
- Sort the array in decreasing order, e.g., using insertion sort;
- return the number at index k-1 in the array.

Algorithm 2 (AlgorithmSortK):

- Store only the first k numbers in an array;
- Sort the array in decreasing order, e.g., using insertion sort;
- Read the rest of the numbers one by one;
    - Ignore if it is smaller than the number at kth position in the array
    - Otherwise; insert it in its correct position in the array and shift the remaining numbers (the number at the kth position will be thrown out of the array)
- Return the number at index k-1 in the array.

3) Input

The program basically takes N +3 numbers as input; first, the *algorithm type*, second, *k*, third, *N*, and then *a total of N numbers*. The program will consume all the inputs from a text file, which contains all the parameters and numbers, each separated by a new line. For example, the contents of the *data.txt* file could be like the following:

```
1
3
5
234
321
324
23
43
```

For this example input file, the algorithm type will be 1 (AlgorithmSortAll). The program should find the 3rd largest element among a list of 5 numbers. These numbers are 234, 321, 324, 23, and 43. See Section 5 for detailed instructions regarding how to read such input numbers from a file.

4) Expected output

The program should print out the kth largest number and the total elapsed time as in the following:

```
Result: 234
Duration(sec): 2
```

5) Testing your program

You have to read the test inputs from a file. The content of the file should conform to the rules described in Section 3. This file should be located in the same folder as the executable program and the name of the file should be provided by executing the program on the console/terminal with the following format:

```
main.exe data.txt
```

In this example, the name of the executable program is `main.exe` and all the test inputs as listed in Section 3 are included in a text file named `data.txt`, which is located in the same folder as `main.exe`.

Your main function should be implemented as in the following to be able to parse the file and obtain its contents via the `cin` command.

```
#include <iostream>
#include <string>
#include <fstream>

...

using namespace std;

int main(int argc, char *argv[]) {

	string testfile;

	if (argc < 2) {
		cout << "Enter a test file name:" << endl;
		cin >> testfile;
	}
	else {
		testfile = argv[1];
	}

	ifstream file(testfile.c_str());
	if (file.is_open()) {
		cin.rdbuf(file.rdbuf());
	}
	else {
		cout << "Error: cannot read the test file!" << endl;
		return -1;
	}

	int algorithmType = 0;
	int k = 0;

	// Numbers are obtained from the file line by line with cin
	cin >> algorithmType;
	cin >> k;

	// Create a TestBed object, initialize and execute the algorithm
	...

	return 0;
}
```

To test your program, you will be provided 5 test cases in the form of text files. Listed below are the name of the text files and the corresponding results expected.

Test input file name	Expected result
test1_500_1000.txt	503
test1_50000_100000.txt	16459
test2_5000_10000.txt	4536
test2_50000_100000.txt	16459
test2_250000_500000.txt	16401

You will also be provided a program that automatically generates test input for your program. The source code of this program (*testInputGenerator.cpp*) will be available so that you can compile it for different platforms, e.g., Windows, Mac-OS, Linux, etc. and obtain an executable program (i.e., `testInputGenerator.exe`). You can run this program on the console/terminal with the following format:

```
testInputGenerator.exe algorithm_type k N number_range > data.txt
```

Hereby, you need to provide concrete values for the command parameters as highlighted with bold fonts. For instance, if you run the program as follows, then you will find the `data.txt` file in the same directory, which includes 1, 3, 5, in the first three lines, followed by 5 numbers that are randomly selected between 0 and 400.

```
testInputGenerator.exe 1 3 5 400 > data.txt
```

6) Submission

You will submit this homework via the LMS system. You should follow the file-naming conventions and guidelines below:

- You should submit your source files as a **ZIP** archive file (**NOT** RAR or other formats). The name of the file should be in format "**<USER-ID>_hw<HOMEWORK-NR>.zip**". For example, if your username is vy1043, then the name of the submitted file should be "vy1043_hw1.zip". Pay attention that all the letters are in lower-case. ZIP archive is supposed to contain **just the source files**, no folders are allowed by any means.
- The contents of the ZIP file should be as follows:
    - **main.cpp** (includes the *main* function)
    - **TestBed.h** (TestBed class definition)
    - **TestBed.cpp** (TestBed class implementation)
    - **SelectionAlgorithm.h** (SelectionAlgorithm class definition)
    - **SelectionAlgorithm.cpp** (SelectionAlgorithm class implementation)
    - **AlgorithmSortK.h** (AlgorithmSortK class definition)
    - **AlgorithmSortK.cpp** (AlgorithmSortK class implementation)
    - **AlgorithmSortAll.h** (AlgorithmSortAll class definition)
    - **AlgorithmSortAll.cpp** (AlgorithmSortAll class implementation)

- Late submissions and C++ files that do not compile are **not** accepted.
- You can resubmit your homework (until the deadline) if you need to.
- Make sure that your program does **not** include commands specific to a development environment, e.g., `system("pause")` or `#pragma once` in Visual Studio.

Hints

A possible approach for the time measurement to be implemented by the method *execute()* in class *TestBed* is shown bellow:

```
#include <ctime>

// Time stamp before the computations
clock_t start = clock();

/* Computations to be measured */

// Time stamp after the computations

clock_t end = clock();

double cpu_time = static_cast<double>( end - start ) /CLOCKS_PER_SEC;
```

A sample code for dynamic allocation of arrays is provided below:

```
int *pNums = 0;     // Define pNums as a pointer to int, initialize to null
int n;              // Define n for keeping the size needed for the array
cin >> n;           // Read in the size
pNums = new int[n];// Allocate n ints and save the pointer in pNums
for (int i=0; i<n; i++) {
    pNums[i] = 0;  // Initialize all the elements to zero
}
delete [] pNums;    // When done, free the memory pointed to by pNums
pNums = 0;          // Clear pNums to prevent using invalid memory reference
```

References

[1] Strategy Design Pattern, https://sourcemaking.com/design_patterns/strategy, [online] accessed in October 2017.