Assignment 7 - Invalid Commands

Due Apr 18 by 11:59pm **Points** 20 **Submitting** a file upload **File Types** h and cpp **Available** until Apr 21 at 12:01am

This assignment was locked Apr 21 at 12:01am.

ST. CHARLES COMMUNITY COLLEGE

CPT-182 - Programming in C++

Programming Assignment - Invalid Commands (20 Points)

(Number in Question Bank: Assignment **7.1**)

Program Overview

In this assignment, you are going to write a class of **dynamic array** of unsigned integers. You **must** use pointers to write the class. Using static arrays or vectors will result in **zero** points for the assignment.

In the **main()** program, you will read some commands from an input file and process them one by one. If a command is **valid**, then process it; if a command is **invalid**, then do **not** process it and write the command ID to the output file.

The Dynamic_Array Class

Before writing any code, you need to review Module **8** again, studying how to write a dynamic array using pointers. Please note that in the lecture, we used a dynamic array of **integers** as example. In this assignment, however, you should write a dynamic array of **unsigned integers**.

- The dynamic array should include the following **private** data fields:
  1) **DEFAULT_CAPACITY**. This is a **const** unsigned integer which has **fixed** value of **10**.
  2) **capacity**. This is an unsigned integer that stores the maximum capacity of the dynamic array.
  3) **num_of_items**. This is an unsigned integer that stores the number of elements stored in the dynamic array. **num_of_items** will always be less than or equal to **capacity**. Please make sure that you understand the difference between **capacity** and **num_of_items**.
  4) **data**. This is a pointer to unsigned integer that stores the elements in the dynamic array.
- The dynamic array should include the following constructors:
  1) **Default constructor**. The default constructor should set **capacity** equal to **DEFAULT_CAPACITY**, set **num_of_items** to **0**, and initialize **data** to a dynamic array of size **capacity**.
  2) **Copy constructor**. You **must** overload the copy constructor. Please review Module **8** again if you are **not** sure how to do it.
- The dynamic array **must** overload the **destructor**. Please review Module **8** again if you are **not** sure how to do it.
- The dynamic array should overload the following operators:
  1) **Deep-copy assignment operator**. Please review Module **8** again if you are **not** sure how to do it.
  2) **Subscript operator** ("[]"). Overload the subscript operator so that the user can access the elements in the dynamic array outside the class using index. Please note that you should overload the operator twice, one for **lvalue** and one for **rvalue**.
- The dynamic array should include the following class-member functions:
  1) **size()**. The function takes **no** arguments and returns **num_of_items**.
  2) **empty()**. The function takes **no** arguments. It returns **true** if the dynamic array is empty; **false** otherwise.
  3) **resize()**. This is a **private** function, taking **no** arguments, and returns **void**. The function doubles the capacity of the dynamic array and keeps the current elements **unchanged**.
  4) **push_back()**. The function takes an unsigned integer as its only argument and returns **void**. The function inserts the value passed in to the rear end of the dynamic array. Do **not** forget to increase the **num_of_items** by **1**.
  5) **pop_back()**. The function takes **no** arguments and returns **void**. The function deletes an element from the rear end of the dynamic array. Do **not** forget to decrease the **num_of_items** by **1**.

For all the class-member functions (including constructors, destructor, and operators), it is your responsibility to correctly determine the return type, number of arguments, types of arguments, whether the function is a **static** function, whether the function is a **friend** function, and whether the function is a **const** function. **Failing** to do these will result in **losing points**.

The Input File

- The input file is a **plain text file** (filename: `commands.txt`).
- Each row in the input file (except those empty lines) stores exactly one command.
- There could be **empty lines** at the beginning, in the middle, and/or at the end of the input file. Your program should smartly skip those empty lines and only process the rows that contain data.
- You **cannot** assume (or guess) the number of commands in the input file. In other words, no matter how many commands are stored in the input file, your program should correctly process all of them.
- There are **2** types of commands, `PUSH` commands and `POP` commands.
- A `PUSH` commands attempts to insert an unsigned integer to the rear end of the dynamic array. Please see below to understand the format of a `PUSH` command. All tokens are separated by whitespaces.

```
    12          PUSH          253
    ↑           ↑             ↑
Command ID  Command Type  Value to Insert
```

- A `POP` command attempts to delete an element from the rear end of the dynamic array. Please see below to understand the format of a `POP` command. All tokens are separated by whitespaces.

```
    6           POP
    ↑           ↑
Command ID  Command Type
```

- There are **2** types of **invalid** commands. Please see the example below (starting with an empty array).

```
0  POP        Invalid. You cannot pop an item from empty array.
1  PUSH  3    Valid. The dynamic array becomes [3].
2  PUSH  -4   Invalid. This is an array of unsigned integers.
3  POP        Valid. The dynamic array becomes empty array.
4  POP        Invalid. You cannot pop an item from empty array.
5  PUSH  15   Valid. The dynamic array becomes [15].
```

- Please refer to the **sample input files** to better understand the input file format.

The Output File

- The output file is a **plain text file** (filename: `invalid.txt`).
- The output file begins with `"INVALID COMMANDS"`. Then, it lists the IDs of all the **invalid** commands in the input file. Each command ID is a separate row in the output file.
- Please refer to the **sample output files** to better understand the output file format.

The `main()` Program

- You are using pointers only in the class of `Dynamic_Array`. In the `main()` program, do **not** use pointers. All you need to do is to create an object of `Dynamic_Array` using its default constructor. Initially, the dynamic array should be empty.
- Your program should read (parse) the commands in the input file one by one. If a command is valid, then apply it to the dynamic array; if it is **invalid**, the do **not** apply it to the dynamic array but write the **command ID** to the output file.
- Since you already took care of the "housekeeping" in the destructor of the dynamic array, you do **not** need to worry about freeing up dynamically allocated memory in the `main()` program.

Sample Input and Output Files (Click to Download)

Sample Input File 1 (https://drive.google.com/uc?export=download&id=1mPj0b2rq7FMOepQR8_h_jV2bSkEjnRfV) **Sample Input File 2**
Sample Output File 1 (https://drive.google.com/uc?export=download&id=1ZGS9W8vRTw3m7n8Sld8CnncQ2MTnTICx) **Sample Output File 2**

Assignment Submission and Grading (Please Read)

- Please upload all your `.h` (if any) and `.cpp` files (**not** the ~~entire Microsoft Visual Studio project folder~~) on Canvas.
- Before the assignment deadline, you can submit your work unlimited times. However, only your latest submission will be graded.
- At least **20**% of your code should be **comments**. All variable, function (if any), and class (if any) names should "make good sense". You should let the grader put **least effort** to understand your code. Grader will **take off points**, even if your program passes all test cases, if he/she has to put extra **unnecessary** effort to understand your code.
- Please **save a backup copy** of all your work in your computer hard drive.

- Your program will be graded (tested) using another valid input file (still named **`commands.txt`**) to check whether it can generate the expected (correct) output file (with correct format and correct output values in it). As long as the input file is valid, your program should generate a correct output file. In other words, your program should work for **any** valid input file, **not** just the sample input files provided in the assignment instructions.
- In this class, you can assume that the input file (input data) is always **valid** and **has correct format**. You do **not** need to deal with ~~invalid input~~ or ~~error handling~~.
- Your work will be graded after the assignment deadline. All students will receive their assignment grades at (almost) the same time.