Assignment 2, Part 1 of 2
Dynamic Stack Implementation

Instructor: Homeyra Pourmohammadali
BME 122- Data Structures and Algorithms
Winter 2021 (Online)
University of Waterloo

Due: 11:59 PM, March 18, 2021

Purpose of this assignment

In this assignment, you will practice your knowledge about **stack** by implementing a data type called **dynamic stack**. You need to use **dynamically re-sizable array** to implement it. (Note that "dynamic" here means that the size of the array can change dynamically. It is different from the meaning of "dynamic" in dynamic memory allocation.) The header file dynamic-stack.h, which is explained below, provides the structure of the DynamicStack class with declarations of member functions. Do not modify the signatures of any of these functions. You need to implement the entire public member functions listed in dynamic-stack.cpp.

Instruction

Sign in to GitLab and verify that you have a project set up for your Assignment 2 (A2) at https://git.uwaterloo.ca/bme122-1211/bme122-1211-<WATIAM_ID> with the following files.

```
YOUR-WATIAM-ID
├── CMakeLists.txt
├── README.md
├── user.yml
├── dynamic-stack.cpp
├── dynamic-stack.h
├── circular-queue.cpp
├── circular-queue.h
└── test.cpp
```

For this part of assignment, you only need to modify dynamic-stack.cpp and user.yml. You can design your own test case and code in test.cpp. It is optional and we will not grade this file.

You can use the same procedures in Assignment 1.1 to pull, edit, build, commit, and push your repo in Gitlab.

Description

The details of the header file dynamic-stack.h are as follows:

Defined using typedef, StackItem is the kind of data that the stack will contain. Being public, it can be accessed directly as DynamicStack::StackItem.

StackItem EMPTY_STACK is a static constant data member that will be used to indicate an empty stack. (Note that any actual data value stored in the stack should not be the same as this value.) Being public, it can be accessed directly as DynamicStack::EMPTY STACK.

Member variables:
items : An array of stack items. (Note that in C++, an array name is a pointer to the first element of the array.)

capacity : Maximum number of elements allowed in the current stack.

size : Current number of elements in the stack.

init capacity : Initial capacity of the array (i.e., the capacity used in the constructor). This is used by pop() to determine if we should decrease the capacity.

Constructors and Destructor:
DynamicStack(): Default constructor of the class DynamicStack. It uses 16 as the initial capacity of the array and allocates the required memory space for the stack. The function appropriately initializes the fields of the created empty stack.

DynamicStack(unsigned int capacity): Parametric constructor of the class Dynamic-Stack. It allocates the required memory space for the stack of the given capacity. The function appropriately initializes the fields of the created empty stack.

~DynamicStack(): Destructor of the class DynamicStack. It deallocates the memory space allocated for the stack.

Constant member functions:
int size() const: Returns the number of items in the stack.

bool empty() const: Returns true if the stack is empty and false otherwise.

void print() const: Prints the stack items sequentially and in order, from the top to the bottom of the stack. (Note: it is mainly used to help you visualize the data. It will not be used in any test cases for grading.)

StackItem peek() const: Returns the value at the top of the stack without removing it. If the stack is empty, it returns the EMPTY STACK constant instead.

Non-constant member functions:
void push(StackItem value): Takes as an argument a StackItem value. If the stack is not full, the value is pushed onto the stack. Otherwise, the capacity of the stack is doubled, and the item is then pushed onto the resized stack.

StackItem pop(): Removes and returns the top item from the stack as long as the stack is not empty. If the number of items remaining in the stack after popping is less than or equal to one quarter of the capacity of the array, then the array is halved. However, if the new halved capacity is less than the initial capacity, then no resizing takes place. Finally, ifthe stack is empty before the pop, the EMPTY STACK constant is returned.

Note

All indexes must start with 0.

Marking

We will try different inputs and check your output. We will only test your program with syntactically and semantically correct inputs.

Part 1 counts 50% of Assignment 2, which is 50 points in total.

Your program runs and does not crash during the test: + 10
Passes Test Cases: + 4 each, in total of 40