BLG252E - Object Oriented Programming

Homework – 1

Assignment Date: 13.03.2017 Due Date : 27.03.2017 at 23:55

In this assignment, you will implement a C++ class called `SmallVector` which is a basic Vector class with small-size optimization.

Vector is a container, which can hold data of arbitrary sizes. The difference between a standard C++ Vector and the `SmallVector` class that you are going to implement in this assignment is that you will avoid dynamic memory allocation for small sized data. Since dynamic memory allocation is a time consuming process compared to static memory allocation, it is generally a good idea to avoid dynamic allocation for small sized data.

Implementation Details

In the class, you will define two buffers: Static Buffer (which is an integer array) and Dynamic Buffer (which is an integer pointer). For this assignment, the static buffer's size should be 32. You should put the given data in the static buffer as long as it is not full. When the static buffer reaches its maximum capacity, you will allocate new memory space to store new values. For example, let's say you need to keep 35 elements (integers) in the vector. Since the size of the static buffer is 32, you should allocate new memory space to store the remaining 3 elements.

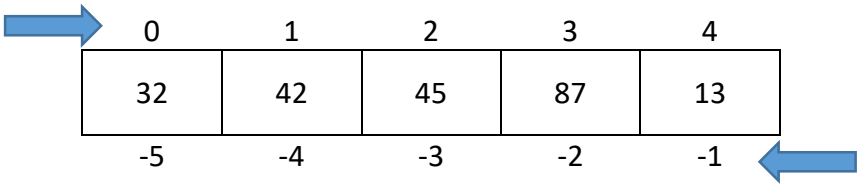
Additionally, you should keep track of the size and capacity of vector. The size of the vector is the number of elements in the vector, while the capacity of the vector is the allocated memory space. For example, assume there are 10 elements in the vector. In this situation, although the vector's size is 10, its capacity is 32 since we can keep 32 elements in the static buffer (as pre-allocated). However, when there is some dynamically allocated memory, you should also take into consideration that memory space for calculating capacity.

You are expected to implement following functionalities:

- **Constructor:** The constructor should optionally take an integer array and its size as parameters. In addition, you are expected to provide a copy constructor.
- **push_back:** takes an element (integer) as parameter and adds it at the end of the `SmallVector`. Furthermore, you need to implement an overloaded version of this function that takes an integer array and its size as parameters.
- **pop_back:** removes the last element from `SmallVector` and returns its value.
- **reverse:** returns current elements in reverse order as a `SmallVector` object.
- **+ operator:** concatenates two `SmallVectors` and returns the resulting `SmallVector`.
- *** operator:** takes an integer and replicates the elements in the `SmallVector` accordingly and returns them as a `SmallVector` object. For example, suppose you

have [0] in a `SmallVector`. Calling ** operator* with 5 as parameter means that it should return a `SmallVector` object that contains five 0's ([0] * 5 = [0,0,0,0,0]). Check the given code below for a better understanding.

- **[] operator:** This operator takes an integer and acts as in arrays for positive numbers. However, for negative numbers, it should act in reverse order. For example, [-2] should return the second last element of a `SmallVector`. You may want to check the figure below for a better understanding.



If the given index number is out of the `SmallVector`'s range, it should return the last element for positive numbers and the first element for negative numbers. Note that, it should be possible to modify the elements of a `SmallVector` by using the subscript operator.

- **= operator:** You need to provide an assignment operator that supports chaining (see your lecture notes).

Please go through the sample main function given below and its output to make sure that your output matches the given output.

```
#include<iostream>
#include"SmallVector.h"

void printSmallVector(const char* const message, const SmallVector& in_vector) {
	std::cout << message << ": ";
	for (int i = 0; i < in_vector.getSize(); i++)
	{
		std::cout << in_vector[i] << " ";
	}
	std::cout << std::endl;
}
int main() {
	int temp[35];
	for (int i = 0; i < 35; i++) {
		temp[i] = i;
	}
	SmallVector test(temp, 10);		// Creating SmallVector with 10 elements
	SmallVector test2(temp, 35);	// Creating SmallVector with 35 elements
	// print test in reverse order
	printSmallVector("Printing test in reverse", test.reverse());
	SmallVector test3; // Empty SmallVector
	test3 = test2 + test; // Concatenate two SmallVectors
	printSmallVector("Printing test3", test3);
	SmallVector test4(test); // Copy SmallVector
	SmallVector test5(temp, 2);
	test5 = test5 * 5; // Replicate elements
	test5.push_back(12); // Add 12 at the end
	test5[0] = 4; // Modify the first element of test5
	std::cout << "Printing test5 in reverse: ";
	for (int i = 1; i < test5.getSize() + 1; i++)
	{
		// Testing negative numbers as indices
		std::cout << test5[-i] << " ";
	}
	return 0;
}
```

```
Printing test in reverse: 9 8 7 6 5 4 3 2 1 0
Printing test3: 0 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 32 33 34 0 1 2 3 4 5 6 7 8 9
Printing test5 in reverse: 12 1 0 1 0 1 0 1 0 1 4
```

Also, you may want to add more test cases to make sure that you wholly implemented the functionality described above.

Hints

- You may want to implement additional private methods to handle memory operations (for expanding and shrinking).
- You can use *memcpy* function to copy data between two memory locations.

Implementation Notes

- Make sure that there is no memory leak in your code. You will ***lose*** a portion of your grade for every memory leak.
- Be careful with the methods/attributes that are supposed to be constant, *static*, *private*/*public*.
- Use comments wherever necessary in your code.
- Please follow a consistent coding style (indentation, variable names, etc.). You can check this link for further information. (This isn't mandatory but it may help you in the future.)
- You are not allowed use STL containers.
- Your program should compile and run on **Linux** environment using **g++ (version 4.8.5 or later).** You can test your program on ITU's Linux Server using SSH protocol**.** Include all necessary header files to your code. Do not use precompiled header files and Windows specific header files and functions.

Submission Notes

- You should put your class declarations and definitions in the "SmallVector.h" and "SmallVector.cpp" files respectively. After that, you should compress into an archive file named "<your_student_number>.zip". Do **NOT** include any executable or project files in the archive file. You should only submit necessary files.
- Submissions are made through the Ninova system and have a ***strict deadline***. Assignments submitted after the deadline will **NOT** be accepted. If you send your homework via e-mail, you will **NOT** get any points. Don't wait until the last minute. Upload whatever you have, you can always overwrite it afterwards.
- This is *not* a group assignment and getting involved in any kind of **cheating** is subject to **disciplinary actions**. Your homework **SHOULD NOT** include any copy-paste material (from the Internet or from someone else's paper/thesis/project). Check the "Academic honesty" section in the syllabus.
- For any questions about the assignment, contact Talha Çolakoğlu via e-mail (colakoglut@itu.edu.tr).