IITM-CS2710 : Programming & Data Structures Lab (Jul-Nov 2018) Lab Session 6

September 10, 2018 **Dynamic Arrays : Vectors** Mode : On Hacker Rank

Objectives:

- Implement your own Vector class with private members and multiple operations.
- Gain a better understanding of how Vectors are implemented internally.
- Practice problem solving using your own vector class.

Overview:

In this assignment, you are expected to write a class called *MyVector* that will facilitate operations defined in Vectors.
so basically, What are Vectors?
Vectors are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically. Vector elements are placed in contiguous storage so that they can be accessed and traversed just like arrays. In vectors, data is inserted at the end and while doing so, sometimes there may be a need of extending the array. Removing the last element can be done directly because no resizing happens.

Detailed Structure of Assignment Task:

This assignment has two components.

Part I [InLab] : Designing the Class "MyVector": We first discuss the design of the structure:

- The `Myvector` class should have three private data members, *size* and *capacity* of type int, *base* of type int *.
  - *size* denotes the actual number of elements present in the vector at a particular instance.
  - *capacity* denotes the maximum number of elements the vector can take at the present configuration.
  - *base* is an array of integers, pointing to the base address of the array.
- The `MyVector` class should have the following `public` functions:
  - `MyVector()`:
    Default constructor, where the initial configuration of the Vector is set.
    This constructor should set the *size* to 0, *capacity* to 1 and allocate space for integers in the *base* array equal to the capacity.
  - `~MyVector()`:
    A destructor to de-allocate the memory allocated to the *base* array

- `void pushback(int val)`:
  - Insert the element *val* at the end of the Vector.
  - If the *size* becomes equal to the *capacity* after insert, resize the vector by doubling its capacity using reserve() function.
- `int popback()`:
  Remove and return the last element from the Vector.
- `void reserve(int newCapacity)`:
  This function is called when *size* = *capacity* which means the initialised vector is full. In this case, it should
  - allocate space equal to the *newCapacity*
  - copy the first *size* number of elements in the newly allocated space
  - free the previous smaller array
  - make the *base* pointer point to this newly allocated space.
- `int size()`:
  Return the number of elements in the Vector.
- `int capacity()`:
  Return the maximum number of elements the Vector can store.
- `bool empty()`:
  Returns *true* if the Vector is empty, else returns $false$.
- `int at(int index)`:
  Returns the element present at the *index* position in the Vector.
- `int front()`:
  Return the first element from the Vector.
- `int back()`:
  Return the last element from the Vector.
- `void clear()`:
  Delete all the elements from the Vector without changing the *capacity*.
- `void display()`:
  Print the contents of the Vector, with each number separated by a space.
- `MyVector & operator= (MyVector & temp)`:
  An Operator Overloading method to assign one MyVector object to another.

- **Note :** Make sure to update the private members after every operation (if required)!
- **Note :** Demonstrate the use of Operator Overloading to your TA for displaying the contents of your Vector once.

Part II [Take Home] : You have been appointed as the new statistician of the IIT-M Cricket Club. IIT-M is scheduled to play *series*-many bilateral series this season with other IITs. The Coach of the team, Kavi Shastri wants the statistics of wicket-takers after each series is played out for the purposes of analyzing his team's strengths and weaknesses. You are tasked with sorting the players on the basis of the number of wickets taken in DESCending order. You need to print the consolidated sorted list after each new series is played out.

For each series, you will get a list of players, their IDs and wickets taken. Some players will be new while others will already be present in your Leaderboard of wicket-takers. After each series, you will be adding new entries or updating old entries and thus at the end of each series, you will be sorting a partially-sorted array. As you had learnt in PDS Class, Insertion Sort works best for online data and when data is partially-sorted, you decide to employ the same here to make your work efficient.You can proceed as follows:

1. Design the Classes *Entry* and *ScoreCard* respectively. The structure of each class is defined below.

2. The *Entry* class should have two private data members, *bowlerID* of unsigned int-type and *wickets* of unsigned int-type. It should have the following public functions:

- `Entry()` : An empty constructor
- `void makeEntry(unsigned int bowlerID,unsigned int wickets)` : A function to initialize the private fields *bowlerID* and *wickets*.
- `unsigned int getID()` : It returns the *bowlerID*.
- `unsigned int getWickets()` : It returns the *wickets* of the bowler.

3. The *ScoreCard* class should have two private data members. *unsignedintseries* that keeps track of the number of series played so far, and a VECTOR of *Entry* objects named *LeaderBoard* that holds the entries of bowlers so far. It should have the following public functions:

- `ScoreCard()` : An constructor that sets Series to 0.
- `void readEntries(unsigned int n)` : Reads the next n-entries from INPUT. If ID is new i.e. it is not already in the *LeaderBoard*, create a new entry. Else update the existing entry.
- `int searchBowler(unsigned int ID)` : It searches for the bowler with given ID in *LeaderBoard*. If found, returns the index of the entry. Else returns -1.
- `void sortEntries()` : It sorts the current set of entries in *LeaderBoard*.If number of wickets are same, then player with smaller ID appears first.
- `void printEntries()` : Prints the current *LeaderBoard* to the OUTPUT.

Input-Output:

Opcode	Corresponding Funtion	Takes Parameter ?	Output ?
1	pushback(int)	Yes	No
2	popback()	No	Yes, print popped element, else -1 if empty
3	size()	No	Yes, print returned size
4	capacity()	No	Yes, print returned capacity
5	empty()	No	Yes, print 1 if empty else 0
6	at(int)	Yes	Yes, print element at the index, else -1 if index out of bounds
7	front()	No	Yes, print returned element
8	back()	No	Yes, print returned element
9	clear()	No	No
10	display()	No	Yes

Part I

Input format

Each test case has the following format:
First line is an integer N, the number of operations that would be performed. N lines follow each of which has one / two integers : $opcode$ and an optional parameter $param$.

Output format

Every opcode that generates an Output, should print it on a new line.
Consider the following testcase :

Sample Input		**Corresponding Output**	**Explanation**
13			
1	52		The first 3 statements just insert 52, 42 then 8
1	42		*Current config : 52 42 8*
1	8		3 returns current size of the vector.
3			5 checks if the vector is empty and prints 0
5			10 displays the contents of the vector
10		3	2 removes and prints the last element
2		0	4 returns the capacity of the vector
4		52 42 8	8 prints the last element
8		8	1 adds element 99 at the end
1	99	4	*Current config : 52 42 99*
6	2	42	6 prints the element at index 2
9		99	9 clears all the elements from the vector
3		0	3 prints the current size

Part II
Input format
Each test case has the following format:

```
series
```

First field is an integer identifier for the number of series IIT-M played. Then *series* many blocks of entries follow where each block of entries for a single series is as follows:

```
B
bowler1 wickets1
bowler2 wickets2
....
bowlerB wicketsB
```

The first number says the number of bowlers who played the series. Then so many entries follow. Each entry has first field as Bowler ID and second field as number of wickets taken by that bowler.
Output format
(Print all entries at the end of each series in descending order of number of wickets taken)

```
Leaderboard after Series:1
bowler1 wickets1
bowler2 wickets2
....
Leaderboard after Series:2
bowler1 wickets1
bowler2 wickets2
bowler3 wickets3
....
.
.
....
Leaderboard after Series:series
bowler1 wickets1
bowler2 wickets2
bowler3 wickets3
....
....
```