### INSTRUCTIONS

**Project 1: Template vector class and iterator**

**Educational Objectives:** Provide practice with a templated vector class, along with practice on the basic concept of an iterator over a container.

**Task:** Implement a templated vector class, along with an associated iterator class for helping with generic container traversals.

**Downloads**

Here are files you will need to finish this assignment.

* [tvector.h](https://www.dropbox.com/s/8ejl38ie0t28iug/tvector.h?dl=0)
* [driver.cpp](https://www.dropbox.com/s/0n1p95obqoymvfs/driver.cpp?dl=0)
* [Links to an external site.](https://www.dropbox.com/s/0n1p95obqoymvfs/driver.cpp?dl=0)
* [driver\_output.txt](https://www.dropbox.com/s/dl8ww62i45xrsp9/driver_output.txt?dl=0)
* [Links to an external site.](https://www.dropbox.com/s/dl8ww62i45xrsp9/driver_output.txt?dl=0)

**Requirements:**

1. NOT using any container classes defined in STL, implement all functions declared in tvector.h in a separate file called tvector.hpp. Note tvector.hpp is included in tvector.h. You may used standard C++ I/O libraries, as well as class libraries like string.
2. Using the tag "-std=c++1", write a Makefile (5pt) and put it in the same folder with tvector.h, tvector.cpp and driver.cpp. The Makefile should compile the driver.cpp driver into an executable called proj1.x. **Your program must be able to compile and run on the linprog machines as this is where TAs will test your code.**
3. Screen out compilation errors. One compilation error costs you 50pt. You will automatically get 0/100pt if there are >= 2 compilation errors.
4. Write **a variety of test cases** to screen out crashing when running proj1.x (10pt). In judging your code, TAs will test on two different input files. If no crash on both test cases, you obtain the 10pt. If your code crashes on both, you only obtain the 5/100pt if your Makefile is correct. Therefore the crash (like segmentation fault) costs you not only these 10pt, but also the 85pt in the main programming part even if you write a few functions correctly. If your program
5. Obtain your result by running driver.cpp. Debug until you get the same result as the provided file driver\_output.txt.

**Submission**:

Tar up your tvector.hpp and Makefile into a single tar archive and submit online via Canvas, in the "Assignments" section.

To do that, copy these two files to a separate folder. Use the linux command "**tar -zcvf tarfilename.tar /path/to/folder/**" in which **/path/to/folder** is the folder which contains only these two files (vector.hpp and Makefile), and **tarfilename.tar** should be lowercase: **lastname\_firstname\_p1.tar**.

Example: to tar a folder whose path is **/mypath/to/folder/**, and generate my submission, I will use this command "**tar -zcvf mallory\_xian\_p1.tar /mypath/to/folder**". You should replace with yours for the tar file and path to folder.

Use the Assignment 1 link to submit this **.tar** file.

To check if you tar your programs correctly, you can use command "**tar -tf tarfilename.tar**" to see if the two files are in your tar file already.

**Information**:

* Define all of the functions for these classes in the file tvector.hpp.
* Always make sure there is ROOM in the array before inserting a new item. If you ever try to insert data, and the array is full (to capacity), then increase the capacity by doubling it.
* Do NOT change any of the prototypes or member data declarations in the tvector.h file at all. All current declarations have the intended prototypes, and they will need to work with my main programs.
* Clean up dynamic memory in appropriate places and do not leave any memory leaks.
* tvector class template

1. The member data of this class consists of a pointer to a dynamically managed array, along with tracking variables for the capacity and the size. Note that the capacity refers to how much space is currently allocated, and size refers to how many data items are currently stored.
2. There is a dummy variable of type T that can be used for error-checking situations. Specifically, whenever a function is to return a stored data item but the container is empty, return (a reference to) the dummy object. Whether you should return the dummy object itself or a reference to it depends on how the function is declared.
3. A static constant called SPARECAPACITY is in the member data so that it can be used to add some extra capacity when constructing vector objects.

* Iterators

1. The small class called TVectorIterator is a helper class that can be used in conjunction with the vector class. This is a common feature used in container classes like this. The purpose of an iterator is to provide a common and non-implementation-specific way of traversing a container, so that mulitple containers could potentially use common algorithms (like sorting and searching functions, for example). For the iterator class in this assignment, here is a brief sample use:

// suppose that V is a vector storing ints, and it has

// already been populated with the values 3, 6, 9, 12, 15, 18, 21

// this call would retrieve a vector iterator over the container V

TVectorIterator<int> itr = V.GetIterator();

// at this point, itr currently is positioned at the first element in

// the list (the 3).

int x = itr.GetValue(); // x would now store 3

itr.Next(); // itr has advanced to the 6

int y = itr.GetValue(); // y would now store 6

itr.Next();

itr.Next(); // we have now advanced to the 12

int z = itr.GetValue(); // z now stores 12

itr.Previous(); // now we have moved backwards, to the 9

int a = itr.GetValue(); // a stores 9. etc.

This class essentially helps us walk through the container in a fairly easy way, with calls to Next() and Previous() to move around.

2. The TVectorIterator class has three member data variables -- a pointer to the data to which it currently refers, the size of the associated vector, and the index from the associated vector. We need data variables for size and index, because the array itself is not inside an iterator object. The pointer will be used to access the data (i.e. it will be needed by the GetData function).

3. The iterator is not intended to be built as a stand-alone item, but rather is created and returned BY member functions in the TVector class, so that it is associated with the vector.

**The followings are the detailed descriptions of the functions that your tvector class template must support, plus the stand-alone operator+ function. (60pt in total)**

* **Default constructor (2pt)** -- creates an empty vector (no data elements), with capacity SPARECAPACITY.
* **TVector(T val, int num) (2pt)** -- creates a vector containing "num" copies of the data element "val", and the capacity should be set to whatever is needed for the stored data, plus the SPARECAPACITY amount.
* **Clear (2pt)** -- clear out the vector so that it represents an empty container (no data elements).
* **Big Five (2pt/each)**
  + Destructor -- appropriate clean-up, no memory leaks
  + Copy constructor -- deep copy
  + Copy assignment operator -- deep copy
  + Move constructor -- constructor with standard move semantics
  + Move assignment operator -- assignment with standard move semantics
* **Accessors (2pt/each)**
  + **IsEmpty** -- returns true if the container is empty, false otherwise
  + **GetSize** -- returns the size (number of data elements)
  + **GetFirst** -- returns the first data element (by reference)
  + **GetLast** -- returns the last data element (by reference)
* Note that error situations in the last two functions would occur if the container was empty (this what the "dummy" item is for).
* **Endpoint insert/removes (4pt/each)**
  + **InsertBack** -- insert the data (parameter) as the last item in the container
  + **RemoveBack** -- remove the last element in the container. If it is empty, just leave it empty
* **Iterator retrieval (2pt/each)**
  + **GetIterator** -- create and return an iterator that is positioned on the first data item in the vector. If empty, return default iterator
  + **GetIteratorEnd** -- create and return an iterator that is positioned on the last data item in the vector. If empty, return default iterator
* **SetCapacity (4pt)**Change the vector's capacity (i.e. grow or shrink it) to the parameter value. IF this would result in a smaller capacity than the current number of data elements, then update the size to match the new capacity. (For example, if there are 10 items in the vector, but the capacity is set to 7, then we will lose the last three data items).
* **Insert (2 parameters) (4pt)**The new data element (second parameter) should be inserted into the vector just *before* the spot referred to by the iterator (the first parameter). If the container is empty, just insert the single item. If the iterator does not refer to a valid spot, then insert at the end of the vector. Function should return an iterator to the newly inserted piece of data. Make sure to update this iterator appropriately before returning, as any change in capacity would change the location of the physical array.
* **Remove (1 parameter) (4pt)**This function should remove the data item that is given by the iterator (the parameter). The function should return an iterator to the next data item (the one that was after the deleted data item). If the initial vector was empty, there's nothing to delete -- so just leave it empty and return a default iterator.
* **Remove (2 parameters) (4pt)**This function should remove the data items in the RANGE that starts at the first iterator (pos1), up through but not including the second iterator (pos2). The function should return an iterator to the next data item (the one that was after the deleted data items). If the initial vector was empty, there's nothing to delete -- so just leave it empty and return a default iterator. If the first iterator is after the second iterator (error situation), don't delete anything, and just return the first iterator (pos1).
* **Print (4pt)**Should print the entire vector contents, front to back, separated by the delimiter given in the second parameter. This function may assume that the stored type T has an available insertion << operator available for printing. Print to the stream given in the first parameter.
* **operator+ (4pt)**This is a standalone function that should return a TVector object that is the result of concatenating two TVector objects together -- in parameter order. See driver.cpp program for examples. (No, this function is NOT intended as a friend function.)

**The followings are the detailed descriptions of the functions that your TVectorIterator class template must support. (3pt/each, 18 points in total)**

* **Default constructor** -- a default iterator should just store the null pointer internally, as well as assume it is referring to an empty vector. We'll call this the "null iterator".
* **HasNext** -- returns true if there is another data element *after* the current one (in the vector), false otherwise
* **HasPrevious** -- returns true if there is another data element *before* the current one, false otherwise
* **Next** -- advances the iterator to the next data element after the current one (unless currently storing nullptr). Returns an iterator to the new position.
* **Previous** -- moves the iterator to the previous data element before the current one (unless currently storing nullptr). Returns an iterator to the new position.
* **GetData** -- return the *data item* at the current iterator position. If the iterator is not pointing to a valid spot (i.e. null pointer), you can use the "dummy" that was defined previously. Note that this is a return by reference.

The last 7 points go to the readability of your code, which includes documentation/comments, appropriate and consistent style/indentation.