Project 2: Vector Class Template Container

Due 02/16/2017

Educational Objectives: Understanding generic programming and information hiding by developing generic containers. Getting familiar with the concept of class template and its usage. Use of iterators, Use of namespace. Operator overloading. Analysis of algorithm complexity.

Statement of Work: Implement a vector class template Vector and its associated iterators. Analyze the complexity of a member function of the developed Vector.

Requirements:

1. A header file Vector.h is provided, which contains the interfaces of the vector class template Vector. In addition to data members and member functions of Vector, both iterator and const_iterator are also defined for Vector. The header file also contains a number of global non-class function templates (overloaded operators). You **cannot** change anything in the Vector.h file.

2. A driver program test_vector.cpp is also provided. It is used to test your implementation of the vector class template for different data types (it tests Vector<int> and Vector<string>). Similarly, you cannot change anything in the test_vector.cpp file. Note that additional tests will be performed to check your implementation of the Vector class template.

3. You need to implement the member functions of the vector class template Vector in a file named Vector.hpp. Note that, Vector.hpp has been included in the header file Vector.h (towards the end of the file). As we have discussed in class, you should not try to compile Vector.hpp (or Vector.h). You should only compile the driver program test_vector.cpp to obtain the executable program. You need to implement all the member functions of Vector class template and non-class global overloaded functions operator==(), operator!=(), and operator<<() included in Vector.h. Vector has three member variables, theSize, theCapacity, and array. theSize records the number of elements currently stored in the vector; theCapacity indicates the maximum number of elements the vector can hold without requesting new memory allocation; and array is a pointer of type T (the memory should be dynamically allocated). The member variable array is used to store elements of the vector. The design of the Vector container closely follows the vector container included in C++/STL, which is also similar to the one presented in the textbook. It is OK for you to adapt the code provided in the textbook. However, you need to note that there are minor differences between the design of the Vector class in this project and the one given given in the textbook. In particular, whenever you need to insert a new element into the vector and the vector is full, you need to double the capacity. If the current capacity is zero, change the new capacity to 1. We describe the requirements of each function in the following.

   Member functions of Vector class template

   - Vector(): Default zero-parameter constructor. This will create an empty vector with both size and capacity to be zero. You need to initialize the member variables. In particular, you need to assign array to NULL (nullptr for c++11).

   - Vector(const Vector &rhs): Copy constructor. Create the new vector using elements in existing vector rhs.

   - Vector(Vector &&rhs): move constructor.

   - Vector(int num, const T & val = T()): Construct a Vector with num elements, all initialized with value val. Note that the capacity should also be num.

- Vector(const_iterator start, const_iterator end): construct a Vector with elements from another Vector between start and end. Including the element referred to by the start iterator, but not by the end iterator, that is, the new vector should contain the elements in the range [start, end).
- ~Vector(): destructor. You should properly reclaim memory.
- operator[](index): index operator. Return reference to the element at the specified location. No error checking on value of index. Note that there are two versions of the index operator.
- operator=(const Vector &rhs): Copy assignment operator
- operator=(Vector &&rhs): move assignment operator
- at(index): Return reference to the element at the specified location. Throw "out_of_range" exception if index is not in the valid range [0, theSize). There are two versions of this member function.
- front() and back(): return reference to the first and last element in the vector, respectively. There are two versions of both member functions.
- size(): return the number of elements currently stored in the vector.
- capacity(): return the number of elements that can be stored in the vector without any new memory allocation.
- empty(): return true if no element is in the vector; otherwise, return false.
- clear(): delete all the elements in the vector. Memory associated with the vector does not need to be reclaimed, put in another way, it will be sufficient if you reset the size of the vector to zero.
- push_back(): insert a new object as the last element into the vector.
- pop_back(): delete the last element in the vector.
- resize(newSize, newValue): Change the size of the vector to newSize. If newSize is greater than the current size theSize, the new positions in the vector should hold the value newValue. Note that capacity may also be changed accordingly.
- reserve(newCapacity): Change the capacity of the vector to newCapacity, if newCapacity is greater than the current size of the vector.
- print(ostream &os, char ofc = ' '): print all elements in the Vector, using character ofc as the deliminator between elements of the vector.
- begin(): return iterator to the first element in the vector. No error checking is required. There are two versions of this member function.
- end(): return iterator to the end marker of the vector (the position after the last element in the vector). No error checking is required. There are two versions of this member function.
- insert(iterator itr, const T & val): insert value val ahead of the element referred by the iterator itr. All current elements in the vector starting at itr should be pushed back by one position. The return value is the iterator referring to the newly inserted element.

- erase(iterator itr): delete element referred to by itr. The return value is the iterator referring to the element following the deleted element.
- erase(iterator start, iterator end): delete all elements between start and end (including start but not end), that is, all elements in the range [start, end). The return value is the iterator referring to the element following the last element being deleted.
- doubleCapacity(): double the capacity of the vector. If the current capacity is 0, set the new capacity to 1. This is a private member function, which can be used by other member functions such as push_back().

Non-class global functions

- operator==(const Vector<T> & lhs, const Vector<T> & rhs): check if two vectors contain the same sequence of elements. Two vectors are equal if they have the same number of elements and the elements at the corresponding position are equal.
- operator!=(const Vector<T> & lhs, const Vector<T> & rhs): opposite of operator==().
- operator<<(ostream & os, const Vector<T> & v): print out all elements in Vector v by calling Vector<T>::print() function.

4. Write a makefile for your project and name your executable as proj2.x. Your program must be able to compile and run on the linprog machines.
5. Analyze the worst-case run-time complexity of the member function erase(iterator itr) of the Vector. Give the complexity in the form of Big-O. Your analysis can be informal; however, it must be clearly understandable by others. Name the file containing the complexity analysis as "analysis.txt".

Downloads

Click here to download the tar file, which contains the following files: Vector.h, test_vector.cpp, and proj2.x. The sample executable program proj2.x was compiled on a linprog machine from test_vector.cpp. As mentioned, you should not compile Vector.h or Vector.hpp directly.

Note: The first person to find a programming error in our program will get a bonus point! (There is no known error in the program.)

Deliverables

Turn in files makefile, Vector.hpp, and analysis.txt in a single tar file via the blackboard.

Hints

Write your own additional test programs to make sure that all public member functions of Vector are tested.