Pros and Cons of Common STL Container Classes

Keith Ecker

University of Wisconsin – Fox Valley

Menasha, Wisconsin

Eckek2561@students.uwc.edu

*ABSTRACT*

STL containers are template classes for common data structures and abstract data types that are implemented through the Standard Library. These containers make it easier for users to implement and traverse these data structures. Each of these containers has specific situations in which they are most useful. It is important to understand these situations and pros and cons of each of these containers in order to better utilize them and increase the quality of programs for users.

I. INTRODUCTION

The Standard Library is a collection of template classes that have been standardized and implemented through the C++ language. This collection is separated into three main sections: containers, algorithms, and iterators. These components work together to allow users to store, traverse, and manipulate data in a standardized way. The containers are class templates that implement common data structures that allows the user to specify the type of the elements that are held within the container. The Standard Library also implements many functions for removing, adding, and manipulating data stored within the container. This library if useful in making it easy for users to implement common abstract data types and raises the level of abstraction in these programs.

II. METHODOLOGY

In this paper the containers are introduced in their own section. In each section a summary of the container along with some functions, applications, time complexity, and code example is given. The pros and cons of each container as well as the importance of knowing these pros and cons is then discussed.

III. STACK

A stack is a container adapter for the standard container deque that is used in a last in, first out context. The Standard Library implements three main function for stacks: push, pop, and top. The push function adds an element to the top of the stack, the pop function removes the top element of a stack, and the top function returns the element that is located on the top of the stack. This means that a stack is a limited access data structure because elements can only be added and removed from the top of the stack only.

stack<int> newStack; //Stack is declared

newStack.push(10); // Stack contents: 10

newStack.push(5); //Stack contents: 5, 10

newStack.push (17); //Stack contents:17, 5, 10

newStack.pop(); //Stack contents: 5,10

new int Top = newStack.top() // Top = 5

Some examples of the applications of the stack data structure are the undo and redo mechanism in text editors, back and forward navigation used in web browsers, and to store activation records of method calls. The time complexity for stack operations for insertion and deletion are constant or O(1) time. This is because the operations of a stack can only access one end of the structure which is the top. The time complexity for accessing and searching are linear or O(n).

IV. QUEUE

A queue is a container adapter for the standard container deque that is used in a first in, first out context. The standard library implements four main functions for queues: push\_back which adds an element to the end of the queue, pop\_front which removes the element at the front of the queue, back which returns the element at the back of the queue, and front which returns the element located at the front of the queue.

queue<int> newQueue; //Queue is declared

newQueue.push(10); // Queue contents: 10

newQueue.push(5); // Queue contents: 10, 5

newQueue.push (17); // Queue contents: 10, 5, 17

newQueue.pop(); // Queue contents: 5, 17

new int Front = newQueue.front() // Front = 5

An example of the applications of the queue are serving requests on shared resource like a printer or CPU. The time complexity for insertion and deletion are constant because elements can only be added and removed from one end of the structure. Elements can only be added to the back and removed from the front. Time complexity for accessing and searching is O(n).

V. VECTOR

Vectors are sequence containers that are dynamic arrays that can resize automatically after adding or removing elements. The Standard Library implements many functions involving iterators, capacity, element access, and modifiers in the vector template class. Elements in vectors can be accessed in a similar way as a static array by using indices. A vector does offer more ways to access elements such as the front and back methods which accesses the front and back elements of the vector.

vector<int> newVector; //Vector is declared

newVector.push\_back(10); // Vector contents: 10

newVector[1] = 5; // Vector contents: 10, 5

new int Front = newVector[0] // Front = 10

new int Back = newVector[1] //Back = 5

Vectors are generally used in place of arrays when the number of elements to be stored is unknown or can increase or decrease at any time. Time complexity for accessing is O(1) while time complexity for searching, insertion, and deletion O(n).

VI. DEQUE

Deques or double ended queues are sequence containers with dynamic sizes that can add and remove elements from the front or back of the data structure. Deques can also randomly access their elements through indices similar to vectors or arrays. The Standard Library implementation for deques share many functions with vectors, but deques have the push front and back methods along with pop back and front. This allows the data structure to add or remove elements from the front or back of the queue.

dequeue<int> new Dequeue; //Dequeue is declared

newDequeue.push\_back(10); //Dequeue contents: 10

newDequeue [1] = 5; // Deque contents: 10, 5

new int Front = newDequeue [0] // Front = 10

new int Back = newDequeue [1] //Back = 5

This is helpful in the “aging” of elements in a deque. The older items are at the front and elements can be removed or added from the front or back of the queue. This is sometimes used in web browser’s history function. The time complexity of access is O(1) while the complexity of search, insertion, and deletion is O(n).

VII. LIST

Lists are sequence containers that link elements together through nodes. These nodes track the address of the next and previous item in the list. The list container is implemented in the same way as a doubly-linked list is. The Standard Library implements many of the same functions that are implemented in the other sequence containers such as insert, push and pop front, and size.

list<int> newList; //List is declared

newList.push\_front(10); // List contents: 10

newList.push\_Front(5); // List contents: 5, 10

newList.push\_back (17); // List contents: 5, 10, 17

newList.pop\_front(); // List contents: 10, 17

new int Front = newList.front() // Front = 10

new int Back = newList.back() //Back = 17

The main upside to using lists is the constant time required to insert and delete items from the data structure. The downside to using lists is that items cannot be accessed directly based on their position in the list. Instead you must iterate from a known position in the list until you reach the desired element. Lists are used to implement many other data structures such as queues, stacks, and trees. One example of the use of lists in a program is a photo gallery application. Lists allow you to iterate over the album by choosing the next or previous photo.

VIII. PRIORITY\_QUEUE

A priority queue is a container adaptor for the standard container vector that has its first element always be either the maximum or minimum based on the strict weak ordering comparison given to the data structure. Priority queue has similar functions as a stack with the push and pop functions, but instead of first in, last out the priority queue can only access the element that is the minimum or maximum based on the ordering given.

priority\_queue<int> newPri; //Priority queue declared

newPri.push(10); //Priority queue contents: 10

newPri.push(5); // Priority queue contents: 10, 5

newPri.push (17); // Priority queue contents:17, 10, 5

newPri.pop(); // Priority queue contents: 10, 5

new int Top = newPri.top() // Top = 10

An application for priority queues is to store a large collection of statistics and return the top items based on a strict weak ordering criterion. The time complexity for accessing elements and searching the priority queue is O(n) and complexity for insertion and deletion is O(log n).

IX. DISCUSSION

Each of these containers have different efficiencies and deficiencies and have different situations in which they are best used for. Stacks are useful when a first in, last out concept is needed like in recursion or backtracking. Deques are useful when you need a queue like data structure that can access elements in constant time through indices. Lists work well when data needs to be stored and accessed in sequential order. Lists are also useful when constant time is needed for insertion and deletion of elements. The queue container can be used when data needs be accessed by a first in, first out basis. An example of this is the jobs that can be queued up for a printer in which the first job to be queued is the first one to be processed and completed. The vector container can be used as dynamically sized arrays in which data can be accessed in constant time through indices. Priority queue is used similar to a queue except the element with the greatest priority is sorted in to the front of the list. An example of the priority queue is a hospital emergency room program which processes the patient with the highest priority first rather than the one that is entered first.

X. CONCLUSSION

In all, each of these containers has unique pros and cons as unique situations in which they are most useful for the user. It is important for these users to understand these pros, cons, and situations so they can better take advantage of each of these containers. When users take advantage of the pros and cons of these containers the overall quality of their programs would increase.

XI. REFRENCES

Rowell, Eric , Et al. "Know Thy Complexities!" Big-O Algorithm Complexity Cheat Sheet. N.p., n.d. Web. 12 Apr. 2017.

*STL Containers*. N.p., n.d. Web. 12 Apr. 2017. <https://msdn.microsoft.com/en-us/library/1fe2x6kt.aspx>.

"STL Containers." *Cplusplus.com*. Cplusplus.com, n.d. Web. 12 Apr. 2017. <http://www.cplusplus.com/reference/stl/>.

Main, M., and Walter J. Savitch. *Data structures & other objects using C*. Boston, MA: Addison-Wesley, 2011. Print.