# Learning C++: The STL and the deque Class

A deque (pronounced like a "deck" of cards) is a double-sided queue where you can add and remove data from either the front of the container or the back of the container. Deques are not a very commonly used container but does have its purposes for a few specialized applications. A deque can be a preferred container when the application requires adding and removing data from both ends of the container.

This article will demonstrate how to use deques and point you to some articles on their use in applications.

## Creating Deques

The deque class is a template class like the other classes in the Standard Template Library (STL). You must import it before you can use it in your programs:

#include <deque>

A deque instance is declared with a data type and a name:

deque<string> words;

deque<int> numbers;

Deques can be initialized with a list:

deque<string> names = {"Cynthia", "Jonathan", "Raymond"};

## Adding Data to a Deque

As I've mentioned, data can be added at the front or at the back of a deque. The functions for this are push\_front and push\_back. Here is an example of these functions at work:

int main()

{

deque<string> names;

names.push\_back("Cynthia");

names.push\_front("Jonathan");

names.push\_back("Danny");

names.push\_front("Raymond");

for (const string s : names) {

cout << s << " "; // Raymond Jonathan Cynthia Danny

}

return 0;

}

I included a range for loop to demonstrate the order in which the elements of the deque are accessed.

## Accessing the Elements of a Deque

I've already demonstrated one way to access the elements of a deque – the range for loop. If you want to access the elements by the indexed position in the deque, you can use the at function:

for (unsigned i = 0; i < names.size(); i++) {

cout << names.at(i) << " ";

}

You can also traverse a deque using an iterator:

for (auto iter = names.begin(); iter != names.end(); iter++) {

cout << \*iter << " ";

}

For more granular access, the functions front and back are used. Here is an example of their use:

int main()

{

deque<string> names;

names.push\_back("Cynthia");

names.push\_front("Jonathan");

names.push\_back("Danny");

names.push\_front("Raymond");

cout << "Front of deque: " << names.front() << endl;

// Raymond

cout << "Back of deque: " << names.back() << endl;

// Danny

return 0;

}

## Removing Deque Elements

The most common way to remove an element from a deque is by calling either the pop\_front function to remove an element from the front of a deque and the pop\_back function for removing an element from the back of a deque. These are the most common removal functions because the primary reason to use a deque is to provide for fast, efficient removal of elements from either the front or the back. If you don't need these efficiencies, you shouldn't be using a deque.

Here is an example of using pop\_front and pop\_back in a program:

int main()

{

deque<string> names;

names.push\_back("Cynthia");

names.push\_front("Jonathan");

names.push\_back("Danny");

names.push\_front("Raymond");

cout << "Removing the front of the deque: " << endl;

names.pop\_front();

cout << "Removing the back of the deque: " << endl;

cout << "The front element is now: "

<< names.front() << endl;

// front is Jonathan

cout << "The back element is now: " << names.back() << endl;

// back is Danny

return 0;

}

As with other containers, you can remove a specific element via an iterator. Here's an example:

int main()

{

deque<string> names;

names.push\_front("Mayo");

names.push\_back("Cynthia");

names.push\_front("Jonathan");

names.push\_back("Danny");

names.push\_front("Raymond");

for (const string s : names) {

cout << s << " ";

}

cout << endl << endl;

string removeElement = "Mayo";

auto iter = names.begin();

while (\*iter != "Mayo") {

iter++;

}

names.erase(iter);

for (const string s : names) {

cout << s << " ";

}

return 0;

}

The output from this program is:

Raymond Jonathan Mayo Cynthia Danny

Raymond Jonathan Cynthia Danny

## Checking for an Empty Deque and Clearing a Deque

The deque class has two utility functions you can use, empty and clear. The empty function checks to see if there is data in a deque, returning either true or false, and the clear function will remove all the data from a deque. Here is a program that uses these two functions:

int main()

{

deque<string> names;

names.push\_front("Mayo");

names.push\_back("Cynthia");

names.push\_front("Jonathan");

names.push\_back("Danny");

names.push\_front("Raymond");

cout << "Size of deque: " << names.size() << endl;

if (!names.empty()) {

names.clear();

}

cout << "Size of deque: " << names.size() << endl;

return 0;

}

## A Sample Application using a Deque

One use for a deque is determining if a word is a palindrome or not. If a word that is formed by pushing its letters into a deque through the back has the same spelling when a new word is formed by popping its letters also from the back, the word is a palindrome.

Here is a program that demonstrates this:

bool checkPal(string word) {

deque<char> letters;

string rword = "";

for (unsigned i = 0; i < word.size(); i++) {

letters.push\_back(word[i]);

}

while (!letters.empty()) {

rword += letters.back();

letters.pop\_back();

}

if (word == rword) {

return true;

}

return false;

}

int main()

{

string aWord;

cout << "Enter a word: ";

cin >> aWord;

if (checkPal(aWord)) {

cout << aWord << " is a palindrome." << endl;

}

else {

cout << aWord << " is not a palindrome." << endl;

}

return 0;

}

Here are two runs from this program:

Enter a word: radar

radar is a palindrome.

Enter a word: hello

hello is not a palindrome.

## Some Possible Applications for Deques

Like a queue, one real use of a deque is in a simulation. One possible simulation is a railroad yard, where there are multiple tracks for railcars to be placed on, but only the tracks on the end of the yard allow railcars to enter and exit. You can simulate this with a deque since a deque only allows input and output from the front or the back, simulating the side tracks of a railroad yard.

Another use of a deque is in keeping track of a web browsing history. New web sites visited are placed at the front of a deque and after some time period, older visited sites are removed from the back of the deque.

The deque is not a frequently used container but it does have it uses and should be considered for applications where data additions and deletions can occur from the front or the back of the container.

Thanks for reading and please email with comments and suggestions.