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

The queue is a container that has first-in, first-out (FIFO) behavior. The classic example of a queue is a grocery store line. Data goes in at the back of a queue and data goes out at the front of a line, with only one data element being able to enter or leave a queue at a time. In this article I'll demonstrate how to use the Standard Template Library's (STL) queue class and discuss some applications where queues are used.

## A Queue Overview

Brits and other Europeans use the word queue when we Americans use the word line. Everyone is familiar with how a line, and hence a queue, works. When you enter a line, you enter the line from the back, unless you are the only person in line, and then you are at both the back and the front of the line.

As people finish conducting their business, they leave the line from the front, and the next person in line moves to the front of the line. When transacting with a line, you normally only conduct business with the front of a line or the back of a line. The people not at the front or the back of a line don't come into play, unless someone tries to "cut" the line, but that's against the rules and I won't be considering that possibility here.

The queue container works the same way. Data enter a queue from the back and leave a queue from the front. You can only examine two positions of a queue – the front and the back. These interface constraints make applications that use queues more efficient and, to be honest, easier to use as there is usually just one way to perform operations.

## Declaring Queues

To use the queue class you must first reference the proper header file:

#include <queue>

The queue container is a template class so you must provide a data type when declaring a new queue. Here are some examples:

queue<string> names;

queue<double> floats;

queue<int> ids;

Queues cannot be initialized with an initializer list and you cannot specify a capacity as a constructor argument.

## Adding Data to Queues and Examing the Front and Back

Data is added to a queue with the push function. Here is an example:

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

int main()

{

vector<string> people = {"Meredith", "Allison", "Mason"};

queue<string> names;

for (const string person : people) {

names.push(person);

}

return 0;

}

The front and back functions are used to examine the front of a queue and the back of a queue respectively. We can add the following code fragment to the program above to see who is in the front of the queue and who is in the back:

cout << "Front of queue: " << names.front() << endl; // Meredith

cout << "Back of queue: " << names.back() << endl; // Mason

## Removing Data from a Queue and Showing the Size of a Queue

The only way to remove data from a queue is with the pop function. This function removes the element at the front of the line, moving all other elements up one position.

You can use the size function to see that an element has been popped from a queue. Here is a program that demonstrates both the pop function and the size function:

int main()

{

vector<string> people = {"Meredith", "Allison", "Mason"};

queue<string> line;

for (const string person : people) {

line.push(person);

}

do {

cout << "Now serving: " << line.front() << endl;

line.pop();

cout << "Number waiting in line: " << line.size() << endl;

} while (!line.empty());

if (line.empty()) {

cout << "Line empty." << endl;

}

return 0;

}

The output from this program is:

Now serving: Meredith

Number waiting in line: 2

Now serving: Allison

Number waiting in line: 1

Now serving: Mason

Number waiting in line: 0

Line empty.

That wraps up my review of the queue class and its functions. Now let's look at some applications for queues.

## Queue Applications

The most common application of queues is in simulations. It's beyond the scope of this article to demonstrate a complex simulation but I can demonstrate a simple simulation - matching partners for a square dance. I am taking this example from the textbook *Data Structures with C++*, by William Ford and William Topp.

This simulation works by placing the men and women participants of the square dance in two separate queues. When it's time to start the dance, partners are formed by taking one man and one woman from the front of each queue. If the two queues do not have the same number of people, the extras must wait until the next dance. To keep things simple, I will get the data (men and women) from two vectors rather than from a file as the textbook example does.

Here’s the program:

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

int main()

{

vector<string> men =

{"George", "Bill", "Bob", "Dave", "Harold", "Dan", "John"};

vector<string> women =

{"Jane", "Sandra", "Shirley", "Louise", "Roberta"};

string man, woman;

queue<string> menDancers, womenDancers;

for (string man : men) {

menDancers.push(man);

}

for (string woman : women) {

womenDancers.push(woman);

}

cout << "Current dance partners are: " << endl << endl;

do {

man = menDancers.front();

woman = womenDancers.front();

menDancers.pop();

womenDancers.pop();

cout << woman << " and " << man << endl;

} while ((!menDancers.empty() && !womenDancers.empty()));

if (!menDancers.empty()){

cout << endl << endl

<< "Men dancers waiting for next dance: "

<< endl << endl;

men.clear();

while (!menDancers.empty()) {

men.push\_back(menDancers.front());

cout << menDancers.front() << endl;

menDancers.pop();

}

}

if (!womenDancers.empty()){

cout << endl << endl

<< "Women dancers waiting for next dance: "

<< endl << endl;

women.clear();

while (!womenDancers.empty()) {

men.push\_back(menDancers.front());

cout << womenDancers.front();

womenDancers.pop();

}

}

return 0;

}

Another example demonstrated by Ford and Topp is the radix sort. [Here](https://en.wikipedia.org/wiki/Radix_sort) is a description of radix sort and how it works. Here is my program for doing radix sort with a queue:

#include <iostream>

#include <queue>

#include <cstdlib>

#include <ctime>

#include <cmath>

using namespace std;

void radixSort(vector<int> &vec, int n) {

queue<int> bins[10];

int maxDigits=3;

int currentDigit=0;

while (currentDigit < maxDigits) {

for(int i=0; i<n; i++){

int divisor=pow(10,currentDigit);

int num = vec[i];

int digitValue = static\_cast<int>((num/divisor)%10);

bins[digitValue].push(num);

}

int i=0;

for(int k=0;k<10;k++){

while (!bins[k].empty()){

int temp=bins[k].front();

vec[i]=temp;

bins[k].pop();

i++;

}

}

currentDigit++;

}

}

int main()

{

const int SZ = 50;

vector<int> numbers;

for (int i = 0; i < SZ; i++) {

numbers.push\_back(rand() % 100 + 1);

}

radixSort(numbers, SZ);

for (int i = 0; i < SZ; i++) {

cout << numbers[i] << " ";

}

return 0;

}

Here is the output from one run of this program:

42 68 35 1 70 25 79 59 63 65 6 46 82 28 62 92 96 43 28 37

1 6 25 28 28 35 37 42 43 46 59 62 63 65 68 70 79 82 92 96

## Other Queue Applications

In many areas of computer science and operations research, queues are used whenever data needs to be held in storage for usage later. In these uses, the queue works as a buffer, in much the same way memory is buffered in computer memory.

Thanks for reading this article and please email me with comments and suggestions.