Searching

0.2.0

Generated by Doxygen 1.8.17

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 myFIFO Class Reference

`#include <fifo.h>`

Public Member Functions

- myFIFO ()
- bool add (int x)
- int remove ()
- void printStats ()
- int lenFull ()
- int bufLen ()
- int getElement (int ii)

3.1.1 Detailed Description

Implements an integer FIFO

Definition at line 18 of file fifo.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 myFIFO()

`myFIFO::myFIFO ( )`

Constructor

Definition at line 17 of file fifo.cpp.

```
17                {
18     for(int ii = 0; ii < bufLength; ii++) {
19         buffer[ii] = 0;
20     }
21 }
```

3.1.3 Member Function Documentation

3.1.3.1 add()

```
bool myFIFO::add (
            int x )
```

Adds a integer to the back of the FIFO

Parameters

x	Integer to add to the FIFO

Returns

true if successful, false otherwise

Definition at line 29 of file fifo.cpp.

```
29                      {
30     //if(bufBack < bufLength) {
31     if(length < bufLength) {
32         buffer[bufBack] = x; // Add value to buffer
33         bufBack++; // equivalent to bufBack = bufBack + 1
34         bufBack = bufBack % bufLength; // wraps around to the beginning
35         length++; // increment length since an element was added
36         return(true);
37     } else {
38         cout « "bufBack exceeded buffer length" « endl;
39         return(false);
40     }
41 }
```

3.1.3.2 bufLen()

```
int myFIFO::bufLen ( )
```

Returns the length of the buffer

Definition at line 103 of file fifo.cpp.

```
103                     {
104     return(bufLength);
105 }
```

3.1.3.3 getElement()

```
int myFIFO::getElement (
            int ii )
```

Returns iith element of the FIFO

Parameters

ii	- which element to return

Definition at line 112 of file fifo.cpp.

```
112                                 {
113     // check ii for invalid values
114     // return the iith element
115     return(buffer[(bufFront + ii) % bufLength]);
116 }
```

3.1.3.4 lenFull()

```
int myFIFO::lenFull ( )
```

Returns the number of full spaces in the fifo

Definition at line 96 of file fifo.cpp.

```
96                          {
97     return(length);
98 }
```

3.1.3.5 printStats()

```
void myFIFO::printStats ( )
```

Prints the information about the buffer

Definition at line 65 of file fifo.cpp.

```
65                          {
66     cout « "------------------" « endl;
67     cout « "bufFront = " « bufFront « " stored at " « &bufFront « endl;
68     cout « "bufBack = " « bufBack « " stored at " « &bufBack « endl;
69 //             cout « "buffer stored at " « buffer « " is:" « endl;
70     cout « "length = " « length « endl;
71 /*
72     // print front
73     for(int ii = 0; ii < bufLength; ii++) {
74         if(ii == bufFront)
75             cout « 'f';
76         cout « '\t';
77     }
78     cout « endl;
79     for(int ii = 0; ii < bufLength; ii++) {
80         cout « buffer[ii] « '\t';
81     }
82     cout « endl;
83     for(int ii = 0; ii < bufLength; ii++) {
84         if(ii == bufBack)
85             cout « 'b';
86         cout « '\t';
87     }
88     cout « endl;
89 */
90     cout « "==================" « endl;
91 }
```

3.1.3.6 remove()

```
int myFIFO::remove ( )
```

Removes an integer from front of the FIFO

Returns

value removed from FIFO, -999999999 if error

Definition at line 48 of file fifo.cpp.

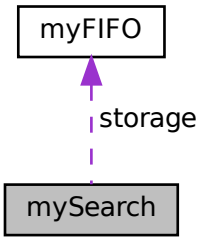
```
48                  {
49     //if(bufFront < bufLength) {
50     if(length > 0) { // bufFront == bufBack means the buffer is empty
51         int retVal = buffer[bufFront];
52         bufFront++;
53         bufFront = bufFront % bufLength;
54         length--; // decrement length since an element was removed
55         return(retVal);
56     } else {
57         cout « "Error tried to remove beyond end of buffer" « endl;
58         return(-999999999);
59     }
60 }
```

The documentation for this class was generated from the following files:

- /home/drseth/CPTR227/20210217SearchClassDemo/src/fifo.h
- /home/drseth/CPTR227/20210217SearchClassDemo/src/fifo.cpp

3.2 mySearch Class Reference

Collaboration diagram for mySearch:



Public Member Functions

- mySearch ()
- void fillStorage (int start)
- void printStorage ()
- int seqSearch (int searchTerm, int &N)
- int binSearch (int searchTerm, int &N)

Public Attributes

- myFIFO storage

  Variable that stores the array.

3.2.1 Detailed Description

Definition at line 16 of file main.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 mySearch()

```
mySearch::mySearch ( )  [inline]
```

Constructor

Definition at line 23 of file main.cpp.

```
23                {
24          cout « "Added a seqSearch instance" « endl;
25      }
```

3.2.3 Member Function Documentation

3.2.3.1 binSearch()

```
int mySearch::binSearch (
             int searchTerm,
             int & N )  [inline]
```

Binary searches for the value passed

This version is based on Malik - Data Structures in C++ 2nd Ed.

Parameters

searchTerm	The term to search for
N	Returns the number of iterations to find searchTerm (Pass by reference)

Returns

Returns the location of searchTerm or -1 if not found

Definition at line 70 of file main.cpp.

```
70                                          {
71         N = 0; // initialize N
72         int first = 0; // index to first item in search area
73         int last = storage.lenFull() - 1; // index to last item in search area
74         int mid; // index to middle item in search area
75         bool found = false; // whether search term has been found
76 
77         while((first <= last) && !found) {
78             N++;
79             mid = (first + last)/2;
80             if(storage.getElement(mid) == searchTerm) {
81                 found = true;
82             } else if(storage.getElement(mid) > searchTerm) {
83                 last = mid - 1;
84             } else {
85                 first = mid + 1;
86             }
87         }
88         if(found) {
89             return(mid);
90         } else {
91             return(-1);
92         }
93     }
```

3.2.3.2 fillStorage()

```
void mySearch::fillStorage (
              int start )  [inline]
```

Fills storage with sequential numbers starting with start

Parameters

start	- The number to start filling at

Definition at line 32 of file main.cpp.

```
32                                  {
33         for(int ii = 0; ii < storage.bufLen(); ii++){
34             storage.add(start++);
35         }
36     }
```

3.2.3.3 printStorage()

```
void mySearch::printStorage ( )  [inline]
```

Definition at line 38 of file main.cpp.

```
38                          {
39         storage.printStats();
40     }
```

3.2.3.4 seqSearch()

```
int mySearch::seqSearch (
              int searchTerm,
              int & N )  [inline]
```

Sequential searches for the value passed

Parameters

searchTerm	The term to search for
N	Returns the number of iterations to find searchTerm (Pass by reference)

Returns

Returns the location of searchTerm or -1 if not found

Definition at line 49 of file main.cpp.

```
49                                              {
50          N = 0; // initialize N
51          for(int ii = 0; ii < storage.lenFull(); ii++) {
52              if(storage.getElement(ii) == searchTerm) {
53                  N = ii+1;
54                  return(ii);
55              }
56          }
57          N = storage.lenFull();
58          return(-1);
59      }
```

3.2.4 Member Data Documentation

3.2.4.1 storage

```
myFIFO mySearch::storage
```

Variable that stores the array.

Definition at line 18 of file main.cpp.

The documentation for this class was generated from the following file:

- /home/drseth/CPTR227/20210217SearchClassDemo/src/main.cpp

Chapter 4

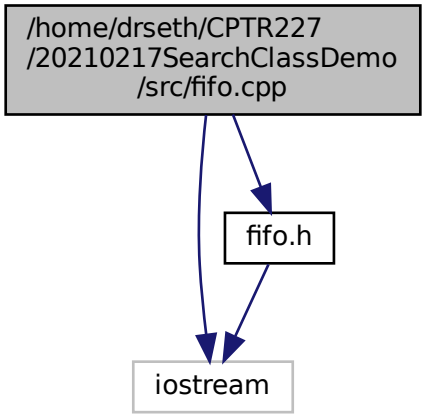
File Documentation

4.1 /home/drseth/CPTR227/20210217SearchClassDemo/src/fifo.cpp File Reference

This is a simple implementation of a FIFO queue.

```
#include <iostream>
#include "fifo.h"
```

Include dependency graph for fifo.cpp:



4.1.1 Detailed Description

This is a simple implementation of a FIFO queue.

This only uses arrays, no STL

Author

Seth McNeill

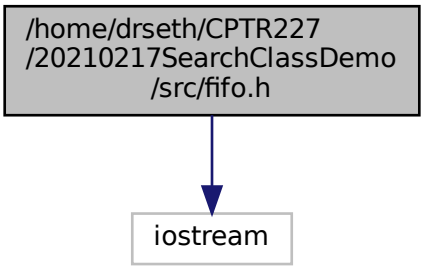
Date

2021 February 02

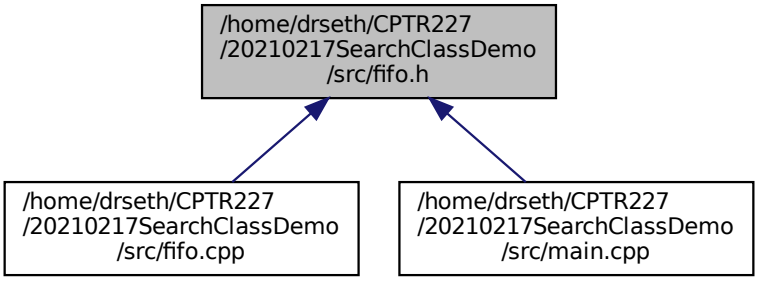
4.2 /home/drseth/CPTR227/20210217SearchClassDemo/src/fifo.h File Reference

This is a simple implementation of a FIFO queue.

`#include <iostream>`
Include dependency graph for fifo.h:



This graph shows which files directly or indirectly include this file:



Classes

- class myFIFO

4.2.1 Detailed Description

This is a simple implementation of a FIFO queue.

This only uses arrays, no STL

Author

Seth McNeill

Date

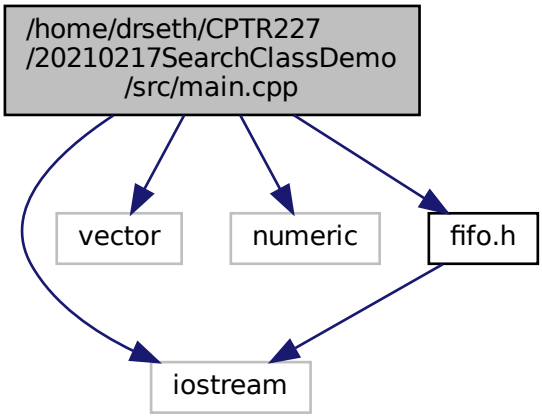
2021 February 02

4.3 /home/drseth/CPTR227/20210217SearchClassDemo/src/main.cpp File Reference

This demonstrates header files, separate cpp files, and some searching.

```
#include <iostream>
#include <vector>
#include <numeric>
#include "fifo.h"
```

Include dependency graph for main.cpp:



Classes

- class mySearch

Functions

- double avg1 (vector< int > const &v)
- int main (int, char **)

4.3.1 Detailed Description

This demonstrates header files, separate cpp files, and some searching.

Implements and times sequential searching using FIFO class

Author

Seth McNeill

Date

2021 February 17

4.3.2 Function Documentation

4.3.2.1 avg1()

```
double avg1 (
            vector< int > const & v )
```

Calculate the average value of a integer vector

This is taken from: https://stackoverflow.com/a/35833470 It uses std::accumulate.

Parameters

v	is a integer std::vector

Returns

The average value of the contents of v

Definition at line 106 of file main.cpp.

```
106                                 {
107     return 1.0 * accumulate(v.begin(), v.end(), 0LL) / v.size();
108 }
```

4.3.2.2 main()

```
int main (
              int ,
              char **  )
```

Definition at line 110 of file main.cpp.

```
110                       {
111     mySearch s1;
112     int nIterations;
113     vector<int> allIters;
114     s1.fillStorage(0);
115     s1.printStorage();
116 /*
117     // need to put internal timing here
118     cout « "Sequential Searching" « endl;
119     for(int ii = 0; ii < (s1.storage.lenFull()+1); ii++)
120     {
121         //cout « "Search for " « ii « " returns ";
122         //cout « s1.search(ii, nIterations) « " in " « nIterations;
123         //cout « " iterations" « endl;
124         s1.seqSearch(ii, nIterations);
125         allIters.push_back(nIterations);
126     }
127     cout « "Calculating the average" « endl;
128     cout « "Sequential average number of iterations is " « avg1(allIters) « endl;
129 */
130     // need to put internal timing here
131     cout « "Binary Searching" « endl;
132     for(int ii = 0; ii < (s1.storage.lenFull()+1); ii++)
133     {
134         //cout « "Search for " « ii « " returns ";
135         //cout « s1.search(ii, nIterations) « " in " « nIterations;
136         //cout « " iterations" « endl;
137         s1.binSearch(ii, nIterations);
138         allIters.push_back(nIterations);
139     }
140     cout « "Calculating the average" « endl;
141     cout « "Binary average number of iterations is " « avg1(allIters) « endl;
142 }
```

Index