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 myHashSearch Class Reference

Public Member Functions

- myHashSearch (int size)
- int modHash (int num)
- int knuthHash (int num)
- bool add (int num)
- void fillStorage (int start)
- void printStorage ()
- int search (int searchTerm, int &N)

Public Attributes

- vector< int > storage

  Variable that stores the array.
- int lenStorage

3.2.1 Detailed Description

This class implements storage and hash search examples

Definition at line 19 of file myHashing.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 myHashSearch()

```
myHashSearch::myHashSearch (
            int size )  [inline]
```

Constructor: Only stores values >=0

Definition at line 27 of file myHashing.cpp.

```
27                          {
28         cout « "Added a seqSearch instance" « endl;
29         lenStorage = size;
30         // create and initialize the storage
31         for(int ii = 0; ii < size; ii++) {
32             storage.push_back(-1);
33         }
34     }
```

3.2.3 Member Function Documentation

3.2.3.1 add()

```
bool myHashSearch::add (
            int num )  [inline]
```

Adds a number to storage using hashing

Parameters

num	the number to add to storage

Returns

true if successful, false otherwise

Definition at line 66 of file myHashing.cpp.

```
66                          {
67         int hashedNum = knuthHash(num);
68         if(storage.at(hashedNum) == -1) { // location is empty
69             storage.at(hashedNum) = num;
70             cout « "success adding " « num « endl;
71             return(true);
72         } else { // location was full
73             // sequential search for empty spot
74             for(int ii = hashedNum+1; ii < lenStorage; ii++) {
75                 if(storage.at(ii) == -1) { //found an empty location
76                     storage.at(ii) = num;
77                     cout « "sequential search added " « num « endl;
78                     return(true);
79                 }
80             }
81             cout « "Collision: failed to add " « num « endl;
82             return(false);
```

```
83         }
84     }
```

3.2.3.2 fillStorage()

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

Fills storage with sequential numbers starting with start

Parameters

start	- The number to start filling at

Definition at line 91 of file myHashing.cpp.

```
91                                  {
92         int ind; // index into the hash table
93         for(int ii = 0; ii < lenStorage; ii++) {
94             cout « "Adding " « start « " at location " « start % lenStorage « endl;
95             storage.at(modHash(start)) = start;
96             start++;
97         }
98     }
```

3.2.3.3 knuthHash()

```
int myHashSearch::knuthHash (
             int num )  [inline]
```

Computes the hash of an integer using a Knuth multiplicative method

Copied from a good stackovervlow page: https://stackoverflow.com/a/665545

Parameters

num	is integer to hash

Returns

hash of num

Definition at line 56 of file myHashing.cpp.

```
56                              {
57         return(num*2654435761 % lenStorage);
58     }
```

3.2.3.4 modHash()

```
int myHashSearch::modHash (
            int num )  [inline]
```

Computes the hash of an integer using the modulus function

Parameters

num	is integer to hash
modNumber	is the integer to modulus by

Returns

hash of num (num % modNumber)

Definition at line 43 of file myHashing.cpp.

```
43                          {
44         return(num % lenStorage);
45     }
```

3.2.3.5 printStorage()

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

prints the contents of storage (beware of calling on large values)

Definition at line 103 of file myHashing.cpp.

```
103                         {
104         if(lenStorage < 10) {
105             for(int ii = 0; ii < lenStorage; ii++) {
106                 cout « storage.at(ii) « '\t';
107             }
108             cout « endl;
109         } else {
110             cout « "Too long to display" « endl;
111         }
112     }
```

3.2.3.6 search()

```
int myHashSearch::search (
            int searchTerm,
            int & N )  [inline]
```

Hash search 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 121 of file myHashing.cpp.

```
121                                         {
122         N = 1; // initialize N
123         int hashedNum = knuthHash(searchTerm);
124         //int hashedNum = modHash(searchTerm);
125 
126         if(storage.at(hashedNum) == searchTerm) {
127             return(hashedNum);
128         } else {
129             // sequential search for empty spot
130             for(int ii = hashedNum+1; ii < lenStorage; ii++) {
131                 N++;
132                 if(storage.at(ii) == searchTerm) { //found the search term
133                     cout << "sequential search found " << searchTerm << " at " << ii << endl;
134                     return(ii);
135                 }
136             }
137             cout << "failed to find " << searchTerm << endl;
138             return(-1);
139         }
140     }
```

3.2.4 Member Data Documentation

3.2.4.1 lenStorage

```
int myHashSearch::lenStorage
```

Definition at line 22 of file myHashing.cpp.

3.2.4.2 storage

```
vector<int> myHashSearch::storage
```

Variable that stores the array.

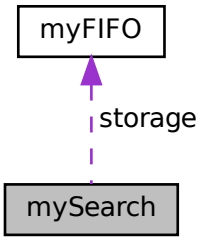
Definition at line 21 of file myHashing.cpp.

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

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

3.3 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.

3.3.1 Detailed Description

This class implements storage and search examples

Definition at line 19 of file main.cpp.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 mySearch()

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

Constructor

Definition at line 26 of file main.cpp.

```
26                {
27         cout « "Added a seqSearch instance" « endl;
28     }
```

3.3.3 Member Function Documentation

3.3.3.1 binSearch()

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

Binary search for the value passed

This requires the data to be ordered in increasing value. This is based off the example in Malik's 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 74 of file main.cpp.

```
74                                          {
75         int first = 0; // index to first value to search
76         int last = storage.lenFull() - 1; // index to last value to search
77         int mid; // index to the middle element
78         bool found = false; // true if searchTerm is found
79         N = 0; // initialize N
80 
81         while((first <= last) && !found) {
82             N++; // increment iteration counter
83             mid = (first + last)/2;
84 
85             if(storage.getElement(mid) == searchTerm) {
86                 found = true;
87             } else if(storage.getElement(mid) > searchTerm) {
88                 last = mid - 1;
89             } else { // searchTerm is > mid->value
90                 first = mid + 1;
91             }
92         }
93         if(found) {
94             return(mid);
95         } else {
96             return(-1);
97         }
98     }
```

3.3.3.2 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 35 of file main.cpp.

```
35                                  {
36          for(int ii = 0; ii < storage.bufLen(); ii++){
37              storage.add(start++);
38          }
39      }
```

3.3.3.3 printStorage()

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

Definition at line 41 of file main.cpp.

```
41                          {
42          storage.printStats();
43      }
```

3.3.3.4 seqSearch()

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

Sequential search 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 52 of file main.cpp.

```
52                                              {
53          N = 0; // initialize N
54          for(int ii = 0; ii < storage.lenFull(); ii++) {
55              if(storage.getElement(ii) == searchTerm) {
56                  N = ii+1;
57                  return(ii);
58              }
59          }
60          N = storage.lenFull();
61          return(-1);
62      }
```

3.3.4 Member Data Documentation

3.3.4.1 storage

```
myFIFO mySearch::storage
```

Variable that stores the array.

Definition at line 21 of file main.cpp.

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

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

Chapter 4

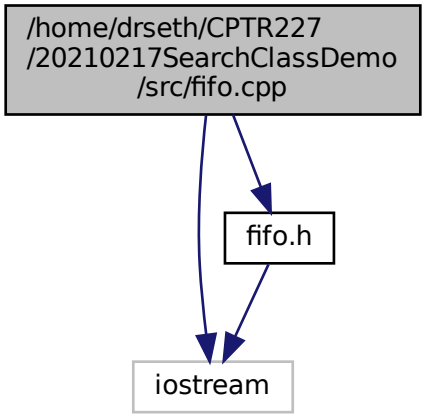
File Documentation

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

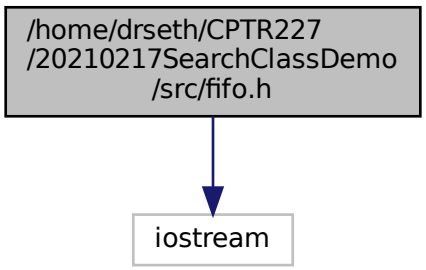
2021 February 02

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

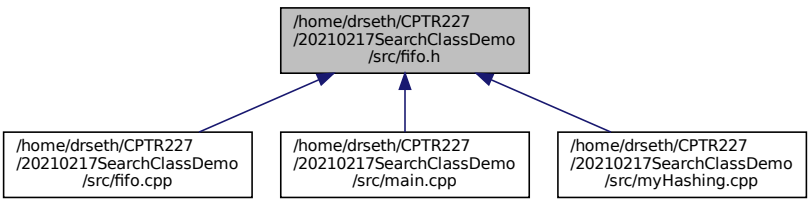
This is a simple implementation of a FIFO queue.

```
#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

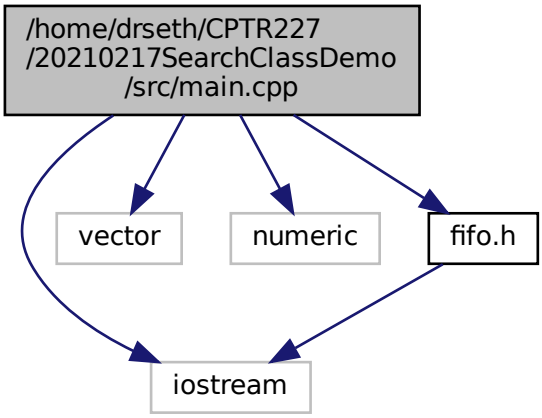
2021 February 02

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 111 of file main.cpp.

```
111                                 {
112     return 1.0 * accumulate(v.begin(), v.end(), 0LL) / v.size();
113 }
```

4.3.2.2 main()

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

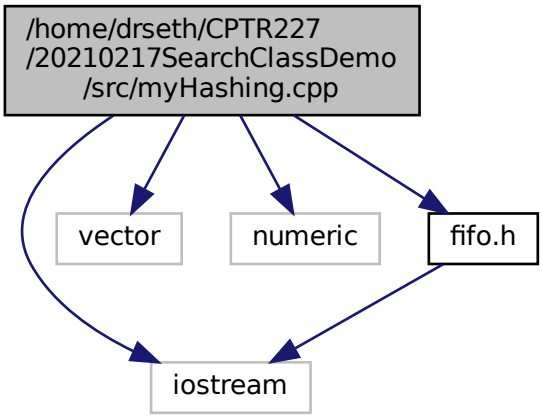
Definition at line 115 of file main.cpp.

```
115                     {
116     mySearch s1;
117     int nIterations;
118     vector<int> allIters;
119     s1.fillStorage(0);
120     s1.printStorage();
121 /*
122     cout « "Sequential searching" « endl;
123     for(int ii = 0; ii < (s1.storage.lenFull()+1); ii++)
124     {
125         //cout « "Search for " « ii « " returns ";
126         //cout « s1.search(ii, nIterations) « " in " « nIterations;
127         //cout « " iterations" « endl;
128         s1.seqSearch(ii, nIterations);
129         allIters.push_back(nIterations);
130     }
131     cout « "Calculating the average" « endl;
132     cout « "The average number of iterations for sequential search is ";
133     cout « avg1(allIters) « endl;
134 */
135     cout « "Binary searching" « endl;
136     for(int ii = 0; ii < (s1.storage.lenFull()+1); ii++)
137     {
138         s1.binSearch(ii, nIterations);
139         allIters.push_back(nIterations);
140     }
141     cout « "Calculating the average" « endl;
142     cout « "The average number of iterations for binary search is ";
143     cout « avg1(allIters) « endl;
144
145 }
```

4.4 /home/drseth/CPTR227/20210217SearchClassDemo/src/myHashing.cpp File Reference

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

Include dependency graph for myHashing.cpp:



Classes

- class myHashSearch

Functions

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

4.4.1 Function Documentation

4.4.1.1 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 154 of file myHashing.cpp.

```
154                                 {
155     return 1.0 * accumulate(v.begin(), v.end(), 0LL) / v.size();
156 }
```

4.4.1.2 main()

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

Definition at line 158 of file myHashing.cpp.

```
158                         {
159     int lenHashTable = 9;
160     myHashSearch s1(lenHashTable);
161     int nIterations;
162     int nTries = 5; // number of items in tryNums
163     int tryNums[] = {11,6,16,21,26};
164     vector<int> allIters;
165 
166     s1.printStorage();
167     for(int ii = 0; ii < nTries; ii++) {
168         s1.add(tryNums[ii]);
169         s1.printStorage();
170     }
171 
172     cout << "Hash based searching" << endl;
173     for(int ii = 0; ii < nTries; ii++)
174     {
175         s1.search(tryNums[ii], nIterations);
176         allIters.push_back(nIterations);
177     }
178     cout << "Calculating the average" << endl;
179     cout << "The average number of iterations for hash search is ";
180     cout << avg1(allIters) << endl;
181 }
```

Index