Lab 1

Manraj Singh
Corey Shimshock
Version 1.0
Mon Feb 8 2021

Table of Contents

Table of Contents

Laboratory 1: Cover Sheet

Names: __Manraj Singh & Corey Shimshock_____ Date _____02/08/2021_____

Section _____902_____

Place a check mark in the *Assigned* column next to the exercises your instructor has assigned to you. Attach this cover sheet to the front of the packet of materials you submit following the laboratory.

Activities	**Assigned:** Check or list exercise numbers	**Completed**
Implementation Testing	✓	✓
Programming Exercise 1	✓	✓
Programming Exercise 2		
Programming Exercise 3		
Analysis Exercise 1	✓	✓
Analysis Exercise 2		
	Total	3

Laboratory 1: Implementation Testing

Names: __Manraj Singh & Corey Shimshock_____ Date _____02/08/2021_____

Section _____902_____

Check with your instructor whether you are to complete this exercise prior to your lab period or during lab.

Test your implementation of the Text ADT using the program in the file *test1.cpp*. This program supports the following tests.

Lab 1 Online Test Plans	
Test	**Action**
1-1	Tests the constructors.
1-2	Tests the length operation.
1-3	Tests the subscript operation.
1-4	Tests the assignment and clear operations.

Test Plan 1-1 (constructors)			
Test case	**String**	**Expected result**	**Checked**
Simple string	`alpha`	`alpha`	✓
Longer string	`epsilon`	`epsilon`	✓
Single-character string	`a`	`a`	✓
Empty string	*empty*	*empty*	✓

Test Plan 1-2 (length operation)			
Test case	**String**	**Expected length**	**Checked**
Simple string	alpha	5	✓
Longer string	epsilon	7	✓
Single-character string	a	1	✓
Empty string	*empty*	0	✓

Test Plan 1-3 (subscript operation)			
Test case	***n***	**Expected character**	**Checked**
Middle character	2	P	✓
First character	0	a	✓
Last character	4	a	✓
Out of range	10	\0	✓

Test Plan 1-4 (assignment and clear operations)			
Test case	**Assignment statement**	**Expected result**	**Checked**
Simple assignment	assignStr = alpha;	alpha	✓
Single-character string	assignStr = a;	a	✓
Empty string	assignStr = empty;	(empty)	✓
Source string longer than destination buffer	assignStr = epsilon;	epsilon	✓
Assign to self	assignStr = assignStr;	epsilon alpha	✓ ✓
Check assignment by clearing destination	assignStr = alpha; assignStr.clear();	(empty)	

Laboratory 1: Programming Exercise 1

Names: __Manraj Singh & Corey Shimshock_____ Date _____02/08/2021_____

Section _____902_____

Test Plan 1-5 (lexical analysis program)		
Test case	**Expected result**	**Checked**
Program in the file *progsamp.dat* void main ( ) { int j , total = 0 ; for (j = 1 ; j <= 20 ; j ++) total += j ; }	[void] [main] [( ] [ )] [{] [int] [j] [,] [total] [=] [0] [;] [for] [( ] [j] [=] [1] [;] [j] [<=] [20] [;] [j] [++] [ )] [total] [+=] [j] [;] [}]	✓ ✓

Laboratory 1: Analysis Exercise 1

Names: _Manraj Singh & Corey Shimshock_____ Date _____02/08/2021_____
Section _____902_____

Part A
What are the implications of having no destructor in a class like Text that does dynamic memory allocation? What are the practical consequences of not having a destructor for these classes in a long-running program?

Dynamic memory allocators are good for supplying class objects with memory without an upper limit, using the term new and utilizing arrays. While the word "new" allocates memory for an object, the word "delete" deallocates it, or wipes the object of memory so it will not unnecessarily increase memory usage. The absence of a destructor is poor for optimizing memory usage and allocation. It is good practice to have a destructor so that once an object is called and does its job in a program run, the destructor frees the memory allocated by that object. Without a destructor in larger, long-running programs, there can be memory leaks, and in cases of looping algorithms, the amount of allocated memory increases with time, and this increases memory usage that does not need to happen. A destructor deallocates memory from class objects once called in each run, so once the algorithm does its job, the memory is wiped, and the program is more optimized and efficient memory-wise.

Part B
What other operators might it make sense to overload in the Text class? Name four and briefly describe how they would work.

1) Overload the assignment operator (=) to initialize one object's value as a copy of another object in the Text class.
2) Cin >> and cout << operators can be overloaded to perform insertion and extraction for a class that can be preset or user defined.
3) The function call operator () can be overloaded for objects of the Text class that can have a different amount of parameters for a function.
4) New and delete[] operators for constructors and destructors can be overloaded to allocate and deallocate memory for Text objects so that the memory usage is not more than needed.

Part C
Are there any operators that it does not make sense to overload in the Text class? Why not?

One such example where it would not make sense to overload in the Text class would be using 'this' or -> when we are dealing with member functions of the Text class. This is because these functions have an implicit 'this' pointer. Due to this, there is no reason to overload -> when dealing with member functions in the Text class.

Class Index

Class List

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

File Index

File List

Here is a list of all files with brief descriptions:

Class Documentation

Text Class Reference

`#include <Text.h>`

Public Member Functions

- **Text** (const char *charSeq="")
- **Text** (const **Text** &other)
- void **operator=** (const **Text** &other)
- **~Text** ()
- int **getLength** () const
- char **operator[]** (int n) const
- void **clear** ()
- void **showStructure** () const

Detailed Description

Definition at line 9 of file Text.h.

Constructor & Destructor Documentation

Text::Text (const char * *charSeq* = "")

Definition at line 12 of file Text.cpp.

Text::Text (const Text & *valueText*)

Preconditions: The sequence must be a char with a numerical value over 0, and the operator must not fail to allocate the requested storage space. Postconditions: The created string will be copied from the charseq, based on the memory allocated buffer. Input: Checks the bufferSize first for buffer creation's memory allocation ability. Output: A copied string is created from a char sequence memory allocator.

Definition at line 32 of file Text.cpp.

Text::~Text ()

Preconditions: If the length will not fit in the buffer, a new buffer array operator is created to assign to buffer. Postconditions: Assigns other to a text object and copies string from other. Input: Input is integer length of bufferSize, which is needed to check for memory allocation validation. Output: A copied string from the verified buffer with length.

Definition at line 65 of file Text.cpp.

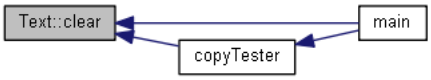
Member Function Documentation

void Text::clear ()

Preconditions: Requested number for index must be greater than or equal to 0, and must be less than the length of the buffer's string. Postconditions: The index of the buffer object character will be returned as requested. Input: From the allocated buffer object, input an integer that is 0 or greater, and less than the length of the string object. Output: Returns the nth character in **Text** object, where the characters are numbered by index.

Definition at line 102 of file Text.cpp.

Here is the caller graph for this function:

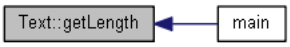


int Text::getLength () const

Preconditions: Destructor must be given memory to be able to free from the **Text** object buffer. Postconditions: The buffer **Text** object has its memory freed. Input: No input, just a buffer array object that had memory allocated will be passed through. Output: A deleted memory allocation for buffer object, so it can be reassigned bufferSize.

Definition at line 76 of file Text.cpp.

Here is the caller graph for this function:



void Text::operator= (const Text & *other*)

Preconditions: Array operator must not fail to allocate requested storage space to allow string to be copied. Postconditions: Copy constructor, creates a copy of valueText after memory allocated. Called whenever: 1) a string is passed to a function using call by value, 2) a function returns a string, or 3) a string is initialized using another string. Input: Starts with the bufferSize, and assigns it buffer object's memory allocation. Output: A copied constructor and valueText.

Definition at line 47 of file Text.cpp.

char Text::operator[] (int *n*) const

Preconditions: The string length must be greater than or equal to 0, and not NULL. Postconditions: The number of characters in the buffer object are returned. Input: Buffer text object gets passed through. Output: Returns the number of characters in the **Text** object buffer besides NULL.

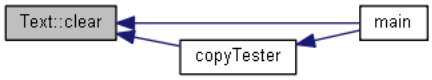
Definition at line 87 of file Text.cpp.

void Text::showStructure () const

Preconditions: Buffer must have valid data stored to be able to clear. Postconditions: The buffer's contents are cleared, but the size remains the same. Input: buffer array object is the input. Output: **Text** object buffer is cleared and emptied, but size unchanged.

Definition at line 113 of file Text.cpp.

Here is the caller graph for this function:



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

- source/repos/Lab 1/Lab 1/**Text.h**
- source/repos/Lab 1/Lab 1/**Text.cpp**

File Documentation

source/repos/Lab 1/Lab 1/Config.h File Reference

Macros

- #define **LAB1_TEST1** 0
- #define **LAB1_TEST2** 0

Macro Definition Documentation

#define LAB1_TEST1 0

Text class (Lab 1) configuration file. Activate test 'N' by defining the corresponding LAB1_TESTN to have the value 1.

Definition at line 6 of file Config.h.

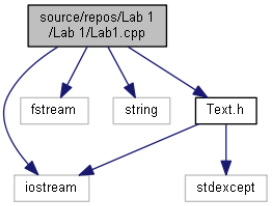
#define LAB1_TEST2 0

Definition at line 7 of file Config.h.

source/repos/Lab 1/Lab 1/Lab1.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include "Text.h"
```

Include dependency graph for Lab1.cpp:



Functions

- void **copyTester** (**Text** copyText)
- void **print_help** ()
- int **main** ()
  Function prototype.

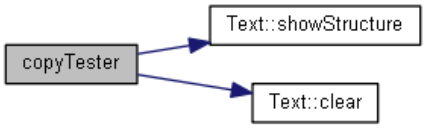
Function Documentation

void copyTester (Text *copyText*)

Preconditions: **Text** class is read in, and objects are created. Postconditions: **Text** obects that were created are ran though various text funtions depending on case Input: User selects which case to run. Output: Various functions of the text class are ran depending on case selected

Definition at line 155 of file Lab1.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

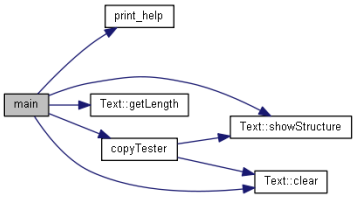
int main ()

Function prototype.

Preconditions: **Text** class is read in, and objects are created. Postconditions: **Text** obects that were created are ran though various text funtions depending on case Input: User selects which case to run. Output: Various functions of the text class are ran depending on case selected

Definition at line 27 of file Lab1.cpp.

Here is the call graph for this function:



void print_help ()

Preconditions: **Text** is pulled from main input and sent through text class function show structure Postconditions: Depending on what is call text is either copied or deleted Input: **Text** from main.cpp depending on case called Output: Either copy of text or clearing of text initially inputed

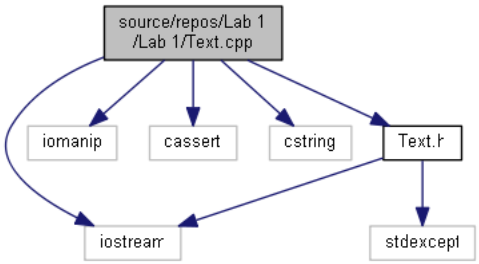
Definition at line 173 of file Lab1.cpp.

Here is the caller graph for this function:

source/repos/Lab 1/Lab 1/Text.cpp File Reference

```
#include <iostream>
#include <iomanip>
#include <cassert>
#include <cstring>
#include "Text.h"
```

Include dependency graph for Text.cpp:




```
9   #include <iostream>
10  #include <fstream>
11  #include <string>
12  #include "Text.h"
13
14
15
16  //-----------------------------------------------------------------
17
18  void copyTester(Text copyText); // copyText is passed by value
19  void print_help();
20  /// Function prototype
21
22  //-----------------------------------------------------------------
23  ///Preconditions: Text class is read in, and objects are created.
24  ///Postconditions: Text obects that were created are ran though various text funtions depending
25  ///Input: User selects which case to run.
26  ///Output: Various functions of the text class are ran depending on case selected
27  int main()
28  {
29
30      Text a("a"), // Predefined test text objects
31          alp("alp"),
32          alpha("alpha").
```

```
33          epsilon("epsilon"),
34          empty,
35          assignText, // Destination for assignment
36          inputText; // Input text object
37      int n; // Input subscript
38      char ch, // Character specified by subscript
39          selection; // Input test selection
40
41          // Get user test selection.
42      print_help();
43
44      cout << "\nPlease enter a selection: ";
45
46      // Execute the selected test.
47      cin >> selection;
48
49      cout << endl;
50      switch (selection)
51      {
52      case '1':
53          // Test 1 : Tests the constructors.
54          cout << "Structure of various text objects: " << endl;
55          cout << "text object: alpha" << endl;
56          alpha.showStructure();
```

```
57          cout << "text object: epsilon" << endl;
58          epsilon.showStructure();
59          cout << "text object: a" << endl;
60          a.showStructure();
61          cout << "empty text object" << endl;
62          empty.showStructure();
63          break;
64      case '2':
65          // Test 2 : Tests the length operation.
66          cout << "Lengths of various text object:" << endl;
67          cout << " alpha   : " << alpha.getLength() << endl;
68          cout << " epsilon : " << epsilon.getLength() << endl;
69          cout << " a       : " << a.getLength() << endl;
70          cout << " empty   : " << empty.getLength() << endl;
71          break;
72
73      case '3':
74          // Test 3 : Tests the subscript operation.
75          cout << "Enter a subscript : ";
76          cin >> n;
77          ch = alpha[n];
78          cout << " alpha[" << n << "] : ";
79          if (ch == '\0')
80              cout << "\\0" << endl;
```

```
81          else
82              cout << ch << endl;
83          break;
84
85      case '4':
86          // Test 4 : Tests the assignment and clear operations.
87          cout << "Assignments:" << endl;
88          cout << "assignText = alpha" << endl;
89          assignText = alpha;
90          assignText.showStructure();
91
92          cout << "assignText = a" << endl;
93          assignText = a;
94          assignText.showStructure();
95
96          cout << "assignText = empty" << endl;
97          assignText = empty;
98          assignText.showStructure();
99
100         cout << "assignText = epsilon" << endl;
101         assignText = epsilon;
102         assignText.showStructure();
103
104         cout << "assignText = assignText" << endl;
```

```
105         assignText = assignText;
106         assignText.showStructure();
107
108         cout << "assignText = alpha" << endl;
109         assignText = alpha;
110         assignText.showStructure();
111
112         cout << "Clear assignText" << endl;
113         assignText.clear();
114         assignText.showStructure();
115
116         cout << "Confirm that alpha has not been cleared" << endl;
117         alpha.showStructure();
118         break;
119
120     case '5':
121         // Test 5 : Tests the copy constructor and operator= operations.
122
123         cout << "Calls by value:" << endl;
124         cout << "alpha before call: " << endl;
125         alpha.showStructure();
126
127         copyTester(alpha);
128         cout << "alpha after call: " << endl;
```

```
129             alpha.showStructure();
130
131             cout << "a before call: " << endl;
132             a.showStructure();
133
134             a = epsilon;
135             cout << "a after call: " << endl;
136             a.showStructure();
137
138             cout << "epsilon after call: " << endl;
139             epsilon.showStructure();
140             break;
141         }
142
143
144
145         system("pause");
146         return 0;
147
148     }
149     ///Preconditions: Text class is read in, and objects are created.
150     ///Postconditions: Text obects that were created are ran though various text funtions depending
151     ///Input: User selects which case to run.
152     ///Output: Various functions of the text class are ran depending on case selected
```

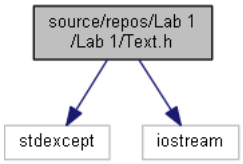
```
153
154     //-----------------------------------------------------------------
155     void copyTester(Text copyText)
156
157     // Dummy routine that is passed a text object using call by value. Outputs
158     // copyText and clears it.
159     {
160         cout << "Copy of text object: " << endl;
161         copyText.showStructure();
162         cout << "Clear copy..." << endl;
163         copyText.clear();
164         copyText.showStructure();
165     }
166     ///Preconditions: Text is pulled from main input and sent through text class function show stru
167     ///Postconditions: Depending on what is call text is either copied or deleted
168     ///Input: Text from main.cpp depending on case called
169     ///Output: Either copy of text or clearing of text initially inputed
170
171     //-----------------------------------------------------------------
172
173     void print_help()
174     {
175         cout << endl << "Tests:" << endl;
176         cout << " 1: Tests the constructors" << endl;
```

```
177         cout << " 2: Tests the length operation" << endl;
178         cout << " 3: Tests the subscript operation" << endl;
179         cout << " 4: Tests the assignment and clear operations" << endl;
180         cout << " 5: Tests the copy constructor and operator= operations" << endl;
181     }
182 ///Preconditions: N/A
183 ///Postconditions: Text is outputed on Screen
184 ///Input: N/A
185 ///Output: Text is outputed on Screen
186
187
188
```

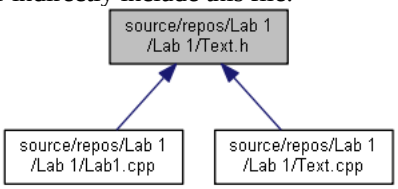
source/repos/Lab 1/Lab 1/Text.h File Reference

```
#include <stdexcept>
#include <iostream>
```

Include dependency graph for Text.h:



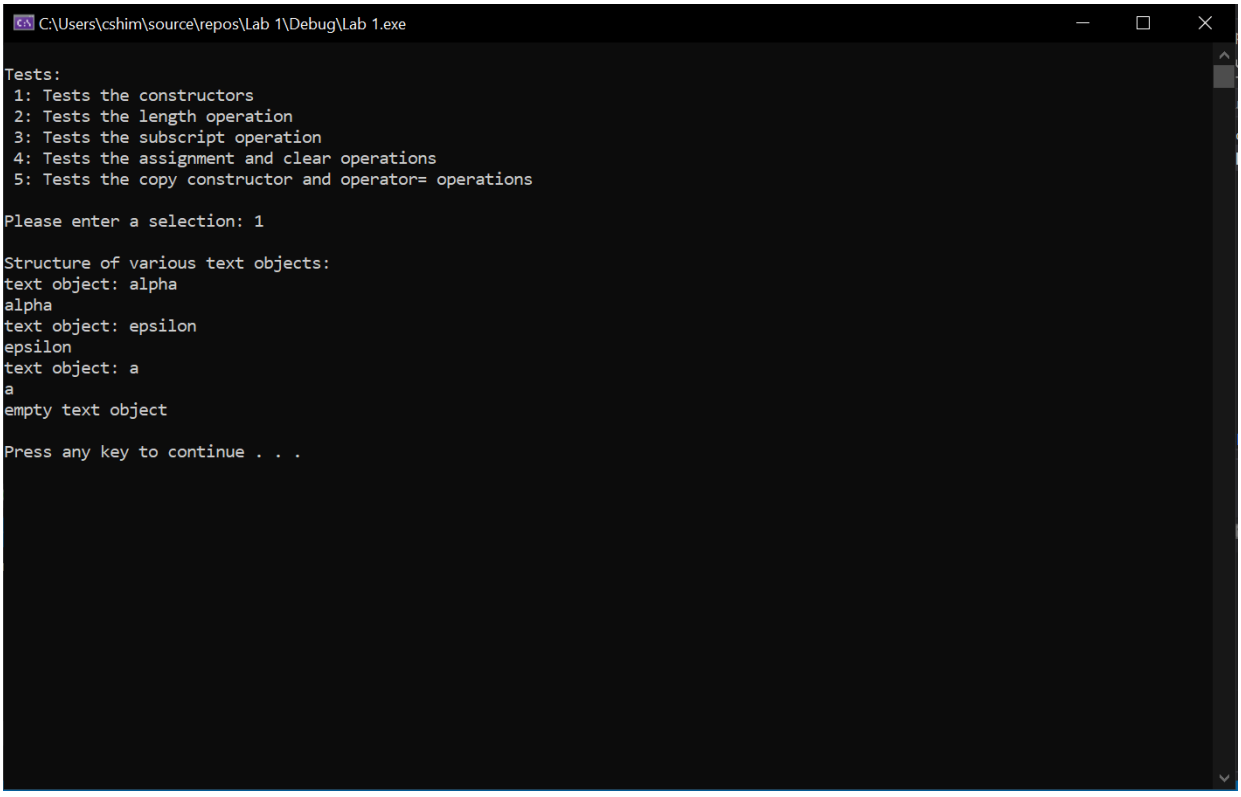
This graph shows which files directly or indirectly include this file:

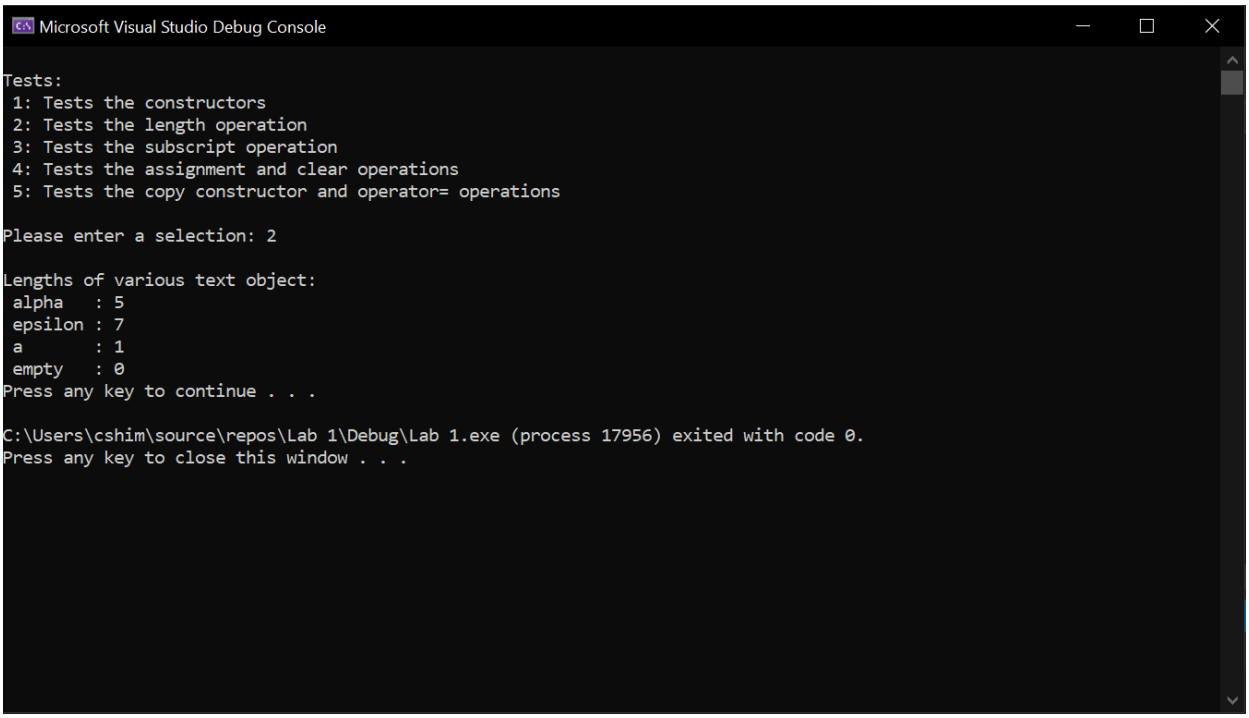


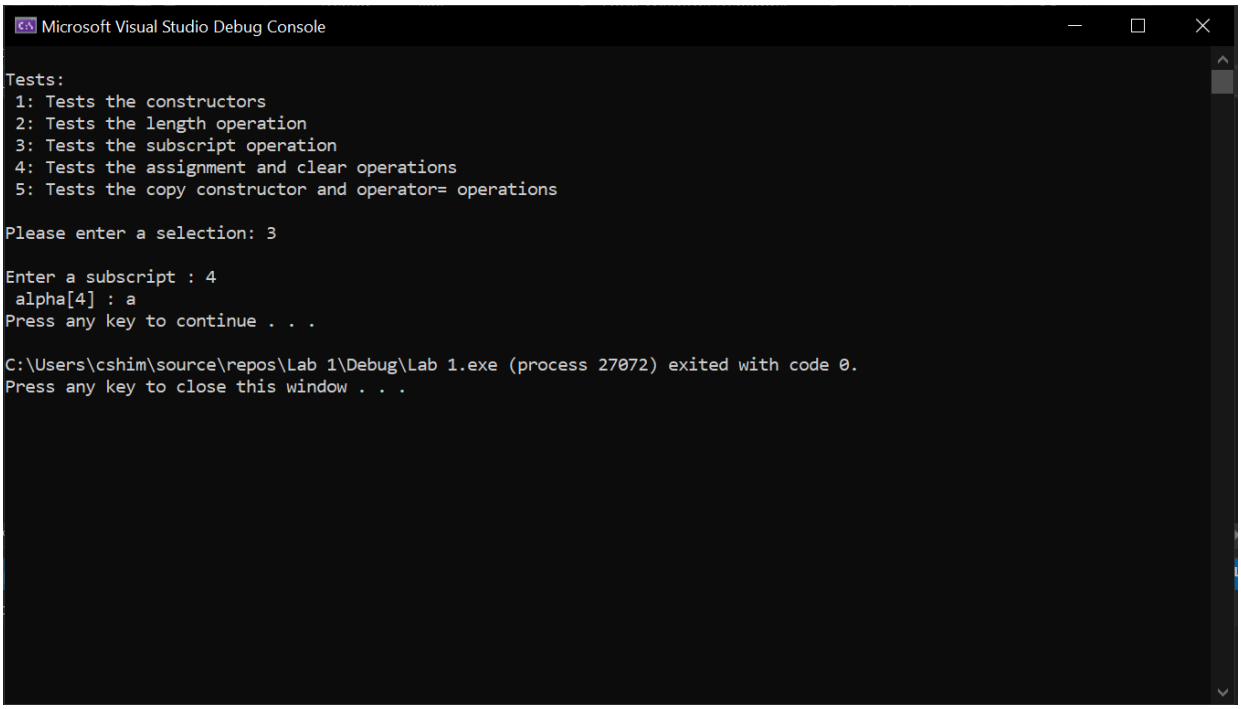
Classes

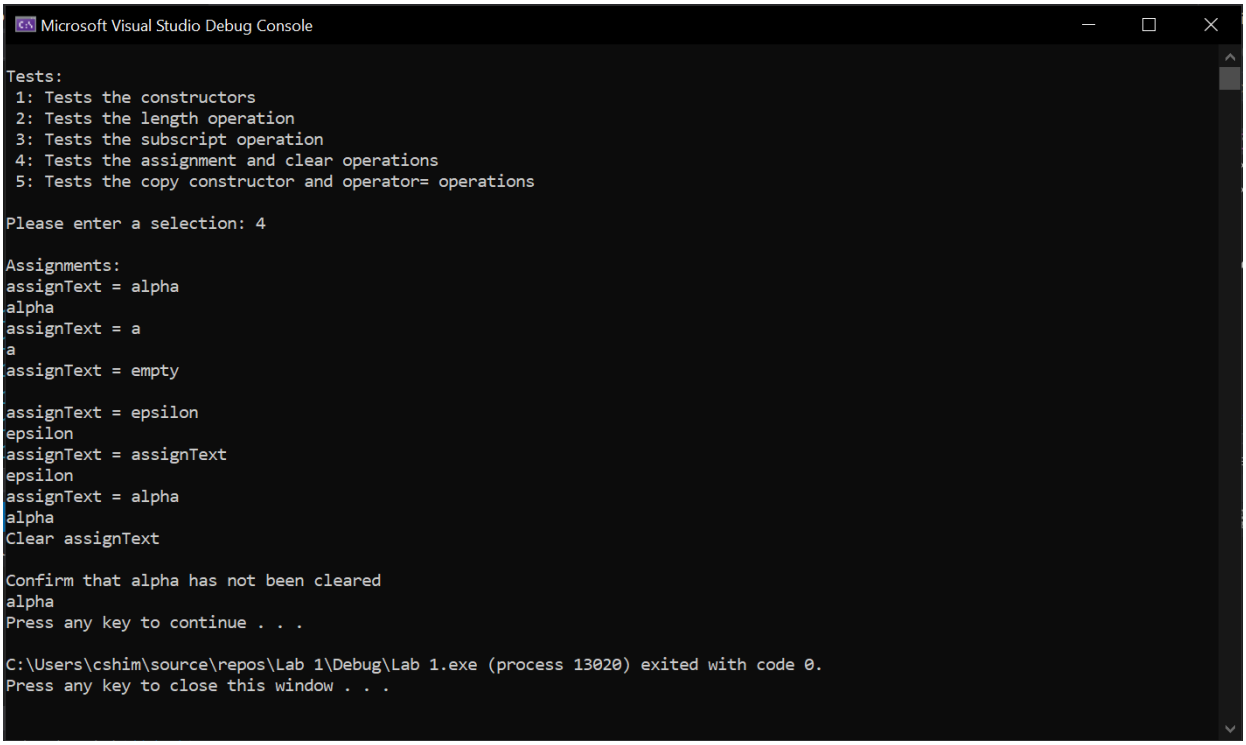
- class **Text**

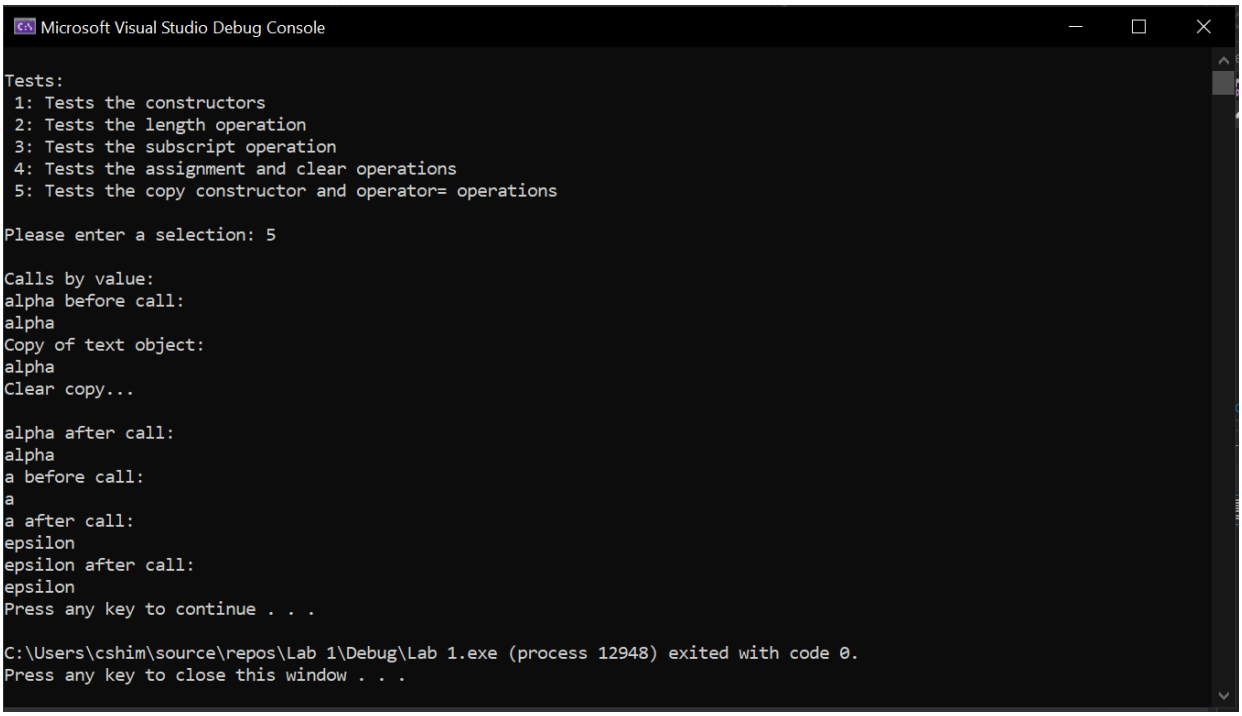
Lab 1 Output


C:\Users\cshim\source\repos\Lab 1\Debug\Lab 1.exe
Tests:
1: Tests the constructors
2: Tests the length operation
3: Tests the subscript operation
4: Tests the assignment and clear operations
5: Tests the copy constructor and operator= operations
Please enter a selection: 1
Structure of various text objects:
text object: alpha
alpha
text object: epsilon
epsilon
text object: a
a
empty text object
Press any key to continue . . .


Microsoft Visual Studio Debug Console
Tests:
1: Tests the constructors
2: Tests the length operation
3: Tests the subscript operation
4: Tests the assignment and clear operations
5: Tests the copy constructor and operator= operations
Please enter a selection: 2
Lengths of various text object:
alpha : 5
epsilon : 7
a : 1
empty : 0
Press any key to continue . . .
C:\Users\cshim\source\repos\Lab 1\Debug\Lab 1.exe (process 17956) exited with code 0.
Press any key to close this window . . .


Microsoft Visual Studio Debug Console
Tests:
1: Tests the constructors
2: Tests the length operation
3: Tests the subscript operation
4: Tests the assignment and clear operations
5: Tests the copy constructor and operator= operations
Please enter a selection: 3
Enter a subscript : 4
alpha[4] : a
Press any key to continue . . .
C:\Users\cshim\source\repos\Lab 1\Debug\Lab 1.exe (process 27072) exited with code 0.
Press any key to close this window . . .


Microsoft Visual Studio Debug Console
Tests:
1: Tests the constructors
2: Tests the length operation
3: Tests the subscript operation
4: Tests the assignment and clear operations
5: Tests the copy constructor and operator= operations
Please enter a selection: 4
Assignments:
assignText = alpha
alpha
assignText = a
a
assignText = empty
assignText = epsilon
epsilon
assignText = assignText
epsilon
assignText = alpha
alpha
Clear assignText
Confirm that alpha has not been cleared
alpha
Press any key to continue . . .
C:\Users\cshim\source\repos\Lab 1\Debug\Lab 1.exe (process 13020) exited with code 0.
Press any key to close this window . . .


Microsoft Visual Studio Debug Console
Tests:
1: Tests the constructors
2: Tests the length operation
3: Tests the subscript operation
4: Tests the assignment and clear operations
5: Tests the copy constructor and operator= operations
Please enter a selection: 5
Calls by value:
alpha before call:
alpha
Copy of text object:
alpha
Clear copy...
alpha after call:
alpha
a before call:
a
a after call:
epsilon
epsilon after call:
epsilon
Press any key to continue . . .
C:\Users\cshim\source\repos\Lab 1\Debug\Lab 1.exe (process 12948) exited with code 0.
Press any key to close this window . . .

File Index

File List

Here is a list of all files with brief descriptions:

Class Documentation

Text Class Reference

`#include <Text.h>`

Public Member Functions

- **Text** (const char *charSeq="")
- **Text** (const **Text** &other)
- void **operator=** (const **Text** &other)
- **~Text** ()
- int **getLength** () const
- char **operator[]** (int n) const
- void **clear** ()
- void **showStructure** () const

Friends

- istream & **operator**>> (istream &input, **Text** &inputText)
- ostream & **operator**<< (ostream &output, const **Text** &outputText)

Detailed Description

Definition at line 14 of file Text.h.

Constructor & Destructor Documentation

Text::Text (const char * *charSeq* = "")

Definition at line 12 of file Text.cpp.

Text::Text (const Text & *valueText*)

Preconditions: The sequence must be a char with a numerical value over 0, and the operator must not fail to allocate the requested storage space. Postconditions: The created string will be copied from the charseq, based on the memory allocated buffer. Input: Checks the bufferSize first for buffer creation's memory allocation ability. Output: A copied string is created from a char sequence memory allocator.

Definition at line 34 of file Text.cpp.

Text::~Text ()

Preconditions: If the length will not fit in the buffer, a new buffer array operator is created to assign to buffer. Postconditions: Assigns other to a text object and copies string from other. Input: Input is integer length of bufferSize, which is needed to check for memory allocation validation. Output: A copied string from the verified buffer with length.

Definition at line 66 of file Text.cpp.

Member Function Documentation

void Text::clear ()

Preconditions: Requested number for index must be greater than or equal to 0, and must be less than the length of the buffer's string. Postconditions: The index of the buffer object character will be returned as requested. Input: From the allocated buffer object, input an integer that is 0 or greater, and less than the length of the string object. Output: Returns the nth character in **Text** object, where the characters are numbered by index.

Definition at line 103 of file Text.cpp.

int Text::getLength () const

Preconditions: Destructor must be given memory to be able to free from the **Text** object buffer. Postconditions: The buffer **Text** object has its memory freed. Input: No input, just a buffer array object that had memory allocated will be passed through. Output: A deleted memory allocation for buffer object, so it can be reassigned bufferSize.

Definition at line 77 of file Text.cpp.

void Text::operator= (const Text & *other*)

Preconditions: Array operator must not fail to allocate requested storage space to allow string to be copied. Postconditions: Copy constructor, creates a copy of valueText after memory allocated. Called whenever: 1) a string is passed to a function using call by value, 2) a function returns a string, or 3) a string is initialized using another string. Input: Starts with the bufferSize, and assigns it buffer object's memory allocation. Output: A copied constructor and valueText.

Definition at line 48 of file Text.cpp.

char Text::operator[] (int *n*) const

Preconditions: The string length must be greater than or equal to 0, and not NULL. Postconditions: The number of characters in the buffer object are returned. Input: Buffer text object gets passed through. Output: Returns the number of characters in the **Text** object buffer besides NULL.

Definition at line 88 of file Text.cpp.

void Text::showStructure () const

Preconditions: Buffer must have valid data stored to be able to clear. Postconditions: The buffer's contents are cleared, but the size remains the same. Input: buffer array object is the input. Output: **Text** object buffer is cleared and emptied, but size unchanged.

Definition at line 114 of file Text.cpp.

Friends And Related Function Documentation

ostream& operator<< (ostream & *output*, const Text & *outputText*) `[friend]`

Preconditions: File stream to be entered Postconditions: output of the input stream Input: **Text** document to be read in Output: States of the input stream

Definition at line 28 of file texio.cpp.

istream& operator>> (istream & *input*, Text & *inputText*) `[friend]`

Definition at line 7 of file texio.cpp.

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

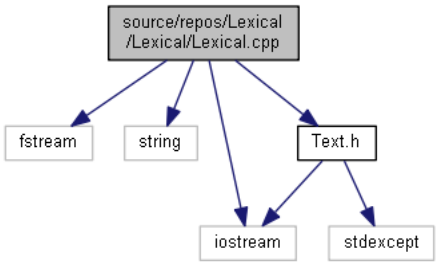
- source/repos/Lexical/Lexical/**Text.h**
- source/repos/Lexical/Lexical/**Text.cpp**

File Documentation

source/repos/Lexical/Lexical/Lexical.cpp File Reference

```
#include <fstream>
#include <string>
#include <iostream>
#include "Text.h"
```

Include dependency graph for Lexical.cpp:



Functions

- int **main** ()

Function Documentation

int main ()

Definition at line 9 of file Lexical.cpp.

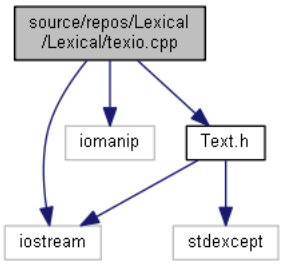
```cpp
#include <fstream>
#include <string>
#include <iostream>
#include "Text.h"
using namespace std;


int main() {
    //set counter to start at 1
    int counter = 1;

    //text token to read each delimited string
    Text token;

    //open text file
    ifstream progFile;
    progFile.open("progsampdat.txt");

    //if statement to validate if file is open
    if (progFile.is_open()) {

        while (!progFile.eof()) {
            //takes in tokens
            progFile >> token;
```


```cpp
#include <fstream>
#include <string>
#include <iostream>
#include "Text.h"
using namespace std;


int main() {
    //set counter to start at 1
    int counter = 1;

    //text token to read each delimited string
    Text token;

    //open text file
    ifstream progFile;
    progFile.open("progsampdat.txt");

    //if statement to validate if file is open
    if (progFile.is_open()) {

        while (!progFile.eof()) {
            //takes in tokens
            progFile >> token;
```

source/repos/Lexical/Lexical/texio.cpp File Reference

```
#include <iostream>
#include <iomanip>
#include "Text.h"
```

Include dependency graph for texio.cpp:



Functions

- istream & **operator**>> (istream &input, **Text** &inputText)
- ostream & **operator**<< (ostream &output, const **Text** &outputText)

Function Documentation

ostream& operator<< (ostream & *output*, const Text & *outputText*)

Preconditions: File stream to be entered Postconditions: output of the input stream Input: **Text** document to be read in Output: States of the input stream

Definition at line 28 of file texio.cpp.

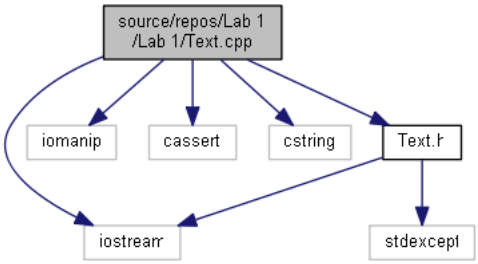
istream& operator>> (istream & *input*, Text & *inputText*)

Definition at line 7 of file texio.cpp.

source/repos/Lexical/Lexical/Text.cpp File Reference

```
#include <iostream>
#include <iomanip>
#include <cassert>
#include <cstring>
#include "Text.h"
```

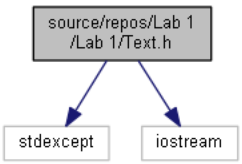
Include dependency graph for Text.cpp:



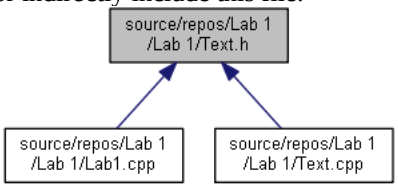
source/repos/Lexical/Lexical/Text.h File Reference

```
#include <stdexcept>
#include <iostream>
```

Include dependency graph for Text.h:



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



Classes

- class **Text**

Lexical Output