

Contents

1	Introduction & Instructions	2
2	Adding More Functionality to the Linked List Class	2
2.1	Linked List Class with a Tail	3
2.1.1	Test Program for Linked List with Tail	5
2.1.2	Program Output	9
2.2	Doubly Linked List Class	12
2.2.1	Test Program for Doubly Linked List	12
2.2.2	Program Output	17
2.3	Extra Credit: Doubly Linked List Class with an Iterator	19
2.3.1	Test Program for Doubly Linked List with Iterator	23
2.3.2	Program Output	31
3	Algorithm and Structure Analysis	39
3.1	Timing Processes in General	39
3.2	Timing Insertions	40
3.3	Timing Sorts	42
4	Application: Scalable Vector Graphics (SVG) Files	44
4.1	Shape Inheritance Hierarchy Description	44
4.2	User Interface Specifications	46
4.2.1	User Interface Specifications Details	47
4.3	SVG Object and File Syntax	51
4.3.1	Rectangle & Square	51
4.3.2	Circle	51
4.3.3	Triangle	52
4.3.4	SVG File Structure	53
4.3.5	Calculating the View Box Values	53
4.4	More Details for the Application	54
4.4.1	User Friendliness	54
4.4.2	Database Data Structure	55

4.4.3	Modularity	55
4.4.4	Memory Management	55
4.5	Extra Credit: Dialog Boxes	55

1 Introduction & Instructions

When you are finished submit all your work through the MyClasses page for this class. Create a directory called Project02, put each programming exercise into its own subdirectory of this directory, zip the entire Project02 directory up into the file Project02.zip, and then submit this zip file to Project #2.

Make sure that you:

- Follow the coding and documentation standards for the course as published in the MyClasses page for the class.
- Check the contents of the zip file before uploading it. Make sure all the files are included.
- Make sure that the file was submitted correctly to MyClasses.

All non-templated class structures are to have their own guarded specification file (.h) and implementation file (.cpp) that has the same name as the class. All templated class structures are to be guarded and written entirely in their (.h) file. No inline coding in the class specification. In addition you must create a make file that compiles and links the project on a Linux computer with a Debian or Debian branch flavor.

This project is set up as a series of exercises centered around linked lists, their function and uses. So it is more of a couple homework/lab assignments instead of a single large application program.

2 Adding More Functionality to the Linked List Class

Here we will be creating two new classes that are built on the linked list class we went over in lecture. The first will add a tail pointer and more convenience functions to the list and the second will be to convert that list into a doubly linked list. There is a third exercise that is for extra credit and it is to additionally add an iterator to the doubly linked list structure to increase the efficiency of the structure.

2.1 Linked List Class with a Tail

For this exercise we will add some functionality to the linked list structure that we developed in class and turn it into a more general use structure.

Our first step is to make the bare bones linked list class we developed from the text into a more useful general storage structure (sometimes referred to as a collection class). Take the templated linked class we developed in lecture, use the third one that has the `ListNode` as an inner class structure and do the following.

1. Change the name of the class to `TLinkedList`. We may be including several different linked list structures in some of these programs and we do not want the names to match. I would also change the name of the file to `TLinkedList.h`, again to keep filenames from clashing.
2. Although we will not be writing any inline code for these structures you may keep the inline code of the constructor of the `ListNode` inner class, for now.
3. Change the name of the `appendNode` function to `push_back`. This is a more common name for the C++ STL operation of appending data to the end of a list.
4. Add in another pointer to a `ListNode` called `tail` to the private section of the `TLinkedList` class. We will of course keep the head pointer to the front of the list, the new tail pointer is to always point to the last node in the linked list. If the list is empty, then the tail pointer, as with the head pointer is to be `nullptr`.
5. Update the `push_back`, `insertNode`, and `deleteNode` functions to correctly keep the tail pointing to the last node in the list. If the list is empty then the tail, as with the head, should be `nullptr`. The `insertNode` and `deleteNode` functions will not change too much but the `push_back` function will be drastically different. Since you have a pointer to the last item you only need to attach the new node to the tail and then adjust the tail to the new node.
6. Add in a copy constructor and overloaded assignment operator.
7. Add in a `push_front` function that will add a new node to the beginning of the list.
8. Add a `pop_front` function that will remove the first node in the list and return the value stored in that node. Here we have a small problem. What if the list is empty? In this case we clearly cannot return a value since no value exists. We have actually encountered this before in different situations but worked around it. We could simply return a default object (type `T`) but there is a problem with this, a default value does not signal an error of any kind and the default value is a legitimate value making it seem like there is an element returned when there was no element to return. This is where the use of exceptions come in handy. If we throw an exception (crash the program) we do not need to return a value from the function and at the same time communicate that an error has occurred. In the main, if we place this function call in

a try-catch block we can also stop the program from crashing. We can use the simplest case of exception handling here to produce exactly what we need. We could be fancier with the exception handling structure but it is really not needed. So if we try to pop the front with an empty list we will throw an exception with string type that simply says `Empty List Exception`.

9. Add a `pop_back` function that will remove the last node in the list and return the value stored in that node. Same setup as `pop_front` function in that a string exception of `Empty List Exception` is returned if the list is empty.
10. Add in a function `peekHead` that takes no parameters and returns the value stored in the first node. In this function, unlike `pop_front`, no nodes are to be deleted from the list. Again we run the risk of trying to peek into an empty list. If the list is empty the function should throw an exception string of `Null Pointer Exception`.
11. Add in a function `peekTail` that takes no parameters and returns the value stored in the last node. In this function no nodes are to be deleted from the list. Again we run the risk of trying to peek into an empty list. If the list is empty the function should throw an exception string of `Null Pointer Exception`.
12. Add in a function `clear` that takes no parameters and removes all the nodes from the list.
13. Add in a function `length` that takes no parameters and returns the number of nodes in the list.
14. Add in a function `get` that takes one integer parameter of the index to return (thinking of the list as an array). The function should return the value of the node at the given index position. Here we run into two possible problems, what if the list is empty and what if the index is outside the bounds of the list? If the list is empty have the function throw an exception with string `Empty List Exception` and if the index is outside the bounds of the array have the function throw an exception with string `Index Out of Bounds Exception`.
15. Add in a function `set` that takes two parameters, an integer parameter of the index to alter (thinking of the list as an array) and the second a value of type `T` to store in the node. The same two possible problems could happen here as with the `get` function, handle them the same way as you did with `get`.
16. Overload the indexing operator `[]`. Again, there are the same considerations here as with `get` and `set`. Handle them in the same manner.
17. Keep the `displayList` function as in the example code.
18. Add in a stream out operator that displays the list contents on one line with commas between the elements and square brackets around the entire list, for example,

```
[7489, 4279, 3010, 987654321, 2352]
```

19. Add in a sort function that will create a new list with the same contents as the original list but in sorted order. As we discussed in class, all we need to do is create an empty list, then for each element in the original list, insert it into the new list. Finally we will return the new list. The original list will be unaltered. So for example, the following will replace list L2 with a sorted version of L1, but L1 will not change.

```
L2 = L1.sort();
```

If we did want to alter L1 to sorted form all we would need to do is,

```
L1 = L1.sort();
```

Make sure you test for memory leaks, very easy to get them when dealing with linked lists. Also, with the indexing operator overloaded you may be tempted to take the lazy way out when writing the copy constructor or overloaded assignment operator. Don't do it. The most efficient way is to traverse the list with pointers. The fewer times you need to move a pointer down the list the faster the execution will be. I will be looking at the efficiency of your code.

2.1.1 Test Program for Linked List with Tail

```
#include <ctime>
#include <iostream>

#include "TLinkedList.h"

using namespace std;

template <class T> void displayHeadTail(TLinkedList<T> &);

void div(string s = "") {
    cout << "\n" << s << "-----\n\n";
}

int main() {
    srand(time(0));
    TLinkedList<int> lst;

    div("Push back and delete tests ");

    lst.push_back(12);
    lst.push_back(4);
    lst.push_back(54);
    lst.push_back(10);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    lst.push_back(101);
    lst.push_back(17);
    lst.push_back(21);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);
}
```

```
lst.deleteNode(12);

lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.deleteNode(101);

lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.deleteNode(1234);

lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.deleteNode(21);

lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.push_back(112);
lst.push_back(14);
lst.push_back(154);
lst.push_back(110);

lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.clear();
displayHeadTail(lst);

div("Insert Tests ");

lst.clear();
lst.insertNode(45);
lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.insertNode(21);
lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.insertNode(57);
lst.displayList();
cout << endl;
displayHeadTail(lst);

for (int i = 0; i < 10; i++)
    lst.insertNode(rand() % 100);

lst.displayList();
cout << endl;
displayHeadTail(lst);

div("Push front tests ");

lst.clear();
```

```
lst.push_front(12);
lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.push_front(15);
lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.push_front(25);
lst.displayList();
cout << endl;
displayHeadTail(lst);

for (int i = 0; i < 10; i++)
    lst.push_front(rand() % 100);

lst.displayList();
cout << endl;
displayHeadTail(lst);

div("Pop front and back with indexing tests ");

try {
    cout << lst.pop_front() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    cout << lst.pop_front() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    cout << lst.pop_front() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    cout << lst.pop_front() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    cout << lst.pop_back() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    cout << lst.pop_back() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    cout << lst.pop_back() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

} catch (string err) {
    cout << err << endl;
}
```

```

try {
    cout << lst[5] << endl;
    cout << lst[2] << endl;
    cout << lst[6] << endl;
    cout << lst[21] << endl;
} catch (string err) {
    cout << err << endl;
}

lst.displayList();
cout << endl;

try {
    lst[5] = 17;
    lst[1] = 29;
    lst[3] = 11;
    lst[52] = 7;
} catch (string err) {
    cout << err << endl;
}

lst.displayList();
cout << endl;
displayHeadTail(lst);

try {
    lst.set(5, -17);
    lst.set(1, -29);
    lst.set(3, -11);
    lst.set(52, -7);
} catch (string err) {
    cout << err << endl;
}

lst.displayList();
cout << endl;
displayHeadTail(lst);

div("CC and = tests ");

TLinkedList<int> L1;
for (int i = 0; i < 5; i++)
    L1.push_back(rand() % 10000 + 1);

TLinkedList<int> L2 = L1;
TLinkedList<int> L3(L1);

L1.displayList();
cout << endl;
L2.displayList();
cout << endl << endl;
L3.displayList();
cout << endl;

L1[3] = 1234567;
L3[3] = 7654321;

L1.displayList();
cout << endl;
L2.displayList();
cout << endl;
L3.displayList();
cout << endl << endl;

```



```

    L3 = L2 = L1;

    L1.displayList();
    cout << endl;
    L2.displayList();
    cout << endl;
    L3.displayList();
    cout << endl << endl;

    L1[3] = 987654321;
    L2[1] = -25;

    L1.displayList();
    cout << endl;
    L2.displayList();
    cout << endl;
    L3.displayList();
    cout << endl << endl;

    L1 = L1;
    L1.displayList();
    cout << endl;

    div("Sort and Stream Tests ");

    cout << L1 << endl;
    cout << L2 << endl;
    cout << L3 << endl;
    cout << endl;

    L1 = L1.sort();
    L2 = L2.sort();
    L3 = L3.sort();

    cout << L1 << endl;
    cout << L2 << endl;
    cout << L3 << endl;

    return 0;
}

template <class T> void displayHeadTail(TLinkedList<T> &L) {
    cout << "Head: ";
    try {
        cout << L.peekHead() << endl;
    } catch (string s) {
        cout << s << endl;
    }

    cout << "Tail: ";
    try {
        cout << L.peekTail() << endl;
    } catch (string s) {
        cout << s << endl;
    }
}

```

2.1.2 Program Output

```

Push back and delete tests -----

12 4 54 10
Head: 12
Tail: 10

```

```
12 4 54 10 101 17 21
Head: 12
Tail: 21
4 54 10 101 17 21
Head: 4
Tail: 21
4 54 10 17 21
Head: 4
Tail: 21
4 54 10 17 21
Head: 4
Tail: 21
4 54 10 17
Head: 4
Tail: 17
4 54 10 17 112 14 154 110
Head: 4
Tail: 110
Head: Null pointer exception.
Tail: Null pointer exception.
```

Insert Tests -----

```
45
Head: 45
Tail: 45
21 45
Head: 21
Tail: 45
21 45 57
Head: 21
Tail: 57
2 21 24 25 30 31 44 45 50 52 57 90 95
Head: 2
Tail: 95
```

Push front tests -----

```
12
Head: 12
Tail: 12
15 12
Head: 15
Tail: 12
25 15 12
Head: 25
Tail: 12
36 97 16 5 69 20 30 0 76 89 25 15 12
Head: 36
Tail: 12
```

Pop front and back with indexing tests -----

```
36
97 16 5 69 20 30 0 76 89 25 15 12
Head: 97
Tail: 12
97
16 5 69 20 30 0 76 89 25 15 12
Head: 16
Tail: 12
16
5 69 20 30 0 76 89 25 15 12
Head: 5
```

```

Tail: 12
5
69 20 30 0 76 89 25 15 12
Head: 69
Tail: 12
12
69 20 30 0 76 89 25 15
Head: 69
Tail: 15
15
69 20 30 0 76 89 25
Head: 69
Tail: 25
25
69 20 30 0 76 89
Head: 69
Tail: 89
89
30
Index Out of Bounds Exception
69 20 30 0 76 89
Index Out of Bounds Exception
69 29 30 11 76 17
Head: 69
Tail: 17
Index Out of Bounds Exception
69 -29 30 -11 76 -17
Head: 69
Tail: -17

```

CC and = tests -----

```

5407 2904 2624 9758 9234
5407 2904 2624 9758 9234

5407 2904 2624 9758 9234
5407 2904 2624 1234567 9234
5407 2904 2624 9758 9234
5407 2904 2624 7654321 9234

5407 2904 2624 1234567 9234
5407 2904 2624 1234567 9234
5407 2904 2624 1234567 9234

5407 2904 2624 987654321 9234
5407 -25 2624 1234567 9234
5407 2904 2624 1234567 9234

5407 2904 2624 987654321 9234

```

Sort and Stream Tests -----

```

[5407, 2904, 2624, 987654321, 9234]
[5407, -25, 2624, 1234567, 9234]
[5407, 2904, 2624, 1234567, 9234]

[2624, 2904, 5407, 9234, 987654321]
[-25, 2624, 5407, 9234, 1234567]
[2624, 2904, 5407, 9234, 1234567]

```

2.2 Doubly Linked List Class

Take the linked list class with tail you constructed in the last exercise and convert it to a doubly linked list. As always the head still points to the first node and tail always points to the last node. In this structure, the ListNode will have two list node pointers, the usual one to the next node and now one to the previous node. The last node's next pointer will be nullptr and the first node's previous pointer will be nullptr as well. We did discuss the possibility of setting up a circular list but we will go that route here. This implementation has the advantage of nullptrs on each end to designate where the list stops.

1. Rename the class DLinkedList, again so that the names do not clash. Also rename the header file DLinkedList.h.
2. Update the displayList function to take in two boolean parameters, the first will determine if the printout of the list is to be vertical or horizontal. True for vertical and false for horizontal, defaulted to false. The second parameter is to designate if the list is to be displayed in forward order or reverse order. True for reverse and false for forward, defaulted to false.
3. Revise all the functions from the singly linked list to work for the doubly linked list. Some functions will not require any alteration but many will. Certainly, any function that is revising the list, adding nodes, or deleting nodes will need to be reworked. Test your work extensively to make sure that all the links are where they need to be, you will probably need to do more testing than just the example testing programs I give you here.
4. Add in a reverse function that will return a new list in reverse order of the current list. The current list should be unaltered. For example,

```
L2 = L1.reverse();
```

will update L2 to be a new list in reverse order of L1, but L1 is not changed by this operation. If we wanted to alter L1 to its reverse order we could simply do the following command.

```
L1 = L1.reverse();
```

5. Make sure that there are no memory errors or leaks.

2.2.1 Test Program for Doubly Linked List

```
#include <ctime>
#include <iostream>

#include "DLinkedList.h"

using namespace std;

template <class T> void displayHeadTail(DLinkedList<T> &);
```

```
void div(string s = "") {
    cout << "\n" << s << "-----\n\n";
}

int main() {
    srand(time(0));
    DLinkedList<int> lst;

    div("Push back and delete tests ");

    lst.push_back(12);
    lst.push_back(4);
    lst.push_back(54);
    lst.push_back(10);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    lst.push_back(101);
    lst.push_back(17);
    lst.push_back(21);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    lst.deleteNode(12);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    lst.deleteNode(101);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    lst.deleteNode(1234);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    lst.deleteNode(21);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    lst.push_back(112);
    lst.push_back(14);
    lst.push_back(154);
    lst.push_back(110);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    div("Print reversed ");

    lst.displayList();
    cout << endl;
```

```
lst.displayList(false, true);
cout << endl;

div("Insert Tests ");
lst.clear();
displayHeadTail(lst);

lst.clear();
lst.insertNode(45);
lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.insertNode(21);
lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.insertNode(57);
lst.displayList();
cout << endl;
displayHeadTail(lst);

for (int i = 0; i < 10; i++)
    lst.insertNode(rand() % 100);

lst.displayList();
cout << endl;
displayHeadTail(lst);

div("Push front tests ");

lst.clear();
lst.push_front(12);
lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.push_front(15);
lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.push_front(25);
lst.displayList();
cout << endl;
displayHeadTail(lst);

for (int i = 0; i < 10; i++)
    lst.push_front(rand() % 100);

lst.displayList();
cout << endl;
displayHeadTail(lst);

div("Pop front and back with indexing tests ");

try {
    cout << lst.pop_front() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    cout << lst.pop_front() << endl;
```

```
lst.displayList();
cout << endl;
displayHeadTail(lst);

cout << lst.pop_front() << endl;
lst.displayList();
cout << endl;
displayHeadTail(lst);

cout << lst.pop_front() << endl;
lst.displayList();
cout << endl;
displayHeadTail(lst);

cout << lst.pop_back() << endl;
lst.displayList();
cout << endl;
displayHeadTail(lst);

cout << lst.pop_back() << endl;
lst.displayList();
cout << endl;
displayHeadTail(lst);

cout << lst.pop_back() << endl;
lst.displayList();
cout << endl;
displayHeadTail(lst);

} catch (string err) {
    cout << err << endl;
}

try {
    cout << lst[5] << endl;
    cout << lst[2] << endl;
    cout << lst[6] << endl;
    cout << lst[21] << endl;
} catch (string err) {
    cout << err << endl;
}

lst.displayList();
cout << endl;

try {
    lst[5] = 17;
    lst[1] = 29;
    lst[3] = 11;
    lst[52] = 7;
} catch (string err) {
    cout << err << endl;
}

lst.displayList();
cout << endl;
displayHeadTail(lst);

try {
    lst.set(5, -17);
    lst.set(1, -29);
    lst.set(3, -11);
    lst.set(52, -7);
} catch (string err) {
```

```
    cout << err << endl;
}

lst.displayList();
cout << endl;
displayHeadTail(lst);

div("CC and = tests ");

DLinkedList<int> L1;
for (int i = 0; i < 5; i++)
    L1.push_back(rand() % 10000 + 1);

DLinkedList<int> L2 = L1;
DLinkedList<int> L3(L1);

L1.displayList();
cout << endl;
L2.displayList();
cout << endl << endl;
L3.displayList();
cout << endl;

L1[3] = 1234567;
L3[3] = 7654321;

L1.displayList();
cout << endl;
L2.displayList();
cout << endl;
L3.displayList();
cout << endl << endl;

L3 = L2 = L1;

L1.displayList();
cout << endl;
L2.displayList();
cout << endl;
L3.displayList();
cout << endl << endl;

L1[3] = 987654321;
L2[1] = -25;

L1.displayList();
cout << endl;
L2.displayList();
cout << endl;
L3.displayList();
cout << endl << endl;

L1 = L1;
cout << L1 << endl;
cout << endl;

div("Reverse tests ");

L2 = L1.reverse();
cout << L1 << endl;
cout << L2 << endl;
cout << endl;

L2.pop_front();
```



```

    L2.pop_back();
    cout << L1 << endl;
    cout << L2 << endl;
    cout << endl;

    div("Sort tests ");

    L2 = L1.sort();
    cout << L1 << endl;
    cout << L2 << endl;
    cout << endl;

    L1 = L1.sort();
    cout << L1 << endl;
    cout << endl;

    return 0;
}

template <class T> void displayHeadTail(DLinkedList<T> &L) {
    cout << "Head: ";
    try {
        cout << L.peekHead() << endl;
    } catch (string s) {
        cout << s << endl;
    }

    cout << "Tail: ";
    try {
        cout << L.peekTail() << endl;
    } catch (string s) {
        cout << s << endl;
    }
}

```

2.2.2 Program Output

Push back and delete tests -----

```

12 4 54 10
Head: 12
Tail: 10
12 4 54 10 101 17 21
Head: 12
Tail: 21
4 54 10 101 17 21
Head: 4
Tail: 21
4 54 10 17 21
Head: 4
Tail: 21
4 54 10 17 21
Head: 4
Tail: 21
4 54 10 17
Head: 4
Tail: 17
4 54 10 17 112 14 154 110
Head: 4
Tail: 110

```

Print reversed -----

```

4 54 10 17 112 14 154 110

```

110 154 14 112 17 10 54 4

Insert Tests -----

Head: Null pointer exception.

Tail: Null pointer exception.

45

Head: 45

Tail: 45

21 45

Head: 21

Tail: 45

21 45 57

Head: 21

Tail: 57

0 14 21 45 49 51 52 53 54 57 61 81 96

Head: 0

Tail: 96

Push front tests -----

12

Head: 12

Tail: 12

15 12

Head: 15

Tail: 12

25 15 12

Head: 25

Tail: 12

70 19 34 3 17 21 23 64 50 25 25 15 12

Head: 70

Tail: 12

Pop front and back with indexing tests -----

70

19 34 3 17 21 23 64 50 25 25 15 12

Head: 19

Tail: 12

19

34 3 17 21 23 64 50 25 25 15 12

Head: 34

Tail: 12

34

3 17 21 23 64 50 25 25 15 12

Head: 3

Tail: 12

3

17 21 23 64 50 25 25 15 12

Head: 17

Tail: 12

12

17 21 23 64 50 25 25 15

Head: 17

Tail: 15

15

17 21 23 64 50 25 25

Head: 17

Tail: 25

25

17 21 23 64 50 25

Head: 17

Tail: 25

```

25
23
Index Out of Bounds Exception
17 21 23 64 50 25
Index Out of Bounds Exception
17 29 23 11 50 17
Head: 17
Tail: 17
Index Out of Bounds Exception
17 -29 23 -11 50 -17
Head: 17
Tail: -17

CC and = tests -----

4260 9038 6717 5417 8489
4260 9038 6717 5417 8489

4260 9038 6717 5417 8489
4260 9038 6717 1234567 8489
4260 9038 6717 5417 8489
4260 9038 6717 7654321 8489

4260 9038 6717 1234567 8489
4260 9038 6717 1234567 8489
4260 9038 6717 1234567 8489

4260 9038 6717 987654321 8489
4260 -25 6717 1234567 8489
4260 9038 6717 1234567 8489

[4260, 9038, 6717, 987654321, 8489]

Reverse tests -----

[4260, 9038, 6717, 987654321, 8489]
[8489, 987654321, 6717, 9038, 4260]

[4260, 9038, 6717, 987654321, 8489]
[987654321, 6717, 9038]

Sort tests -----

[4260, 9038, 6717, 987654321, 8489]
[4260, 6717, 8489, 9038, 987654321]

[4260, 6717, 8489, 9038, 987654321]

```

2.3 Extra Credit: Doubly Linked List Class with an Iterator

This exercise is not required for the project. It is adding an iterator like structure to the doubly linked list from the previous exercise. Although I would like it if everyone tried this exercise it does have details that can be a bit frustrating and difficult to implement and debug. The concept is fairly easy and the implementation is really not that bad but it is also easy to code yourself into a corner that may be difficult to fix. So give yourself some time and patience if you give this a try.

I said above that this will be an “iterator like” structure to our list. C++ defines the specific criteria for a full iterator structure, you can find these criteria online.

<https://en.cppreference.com/w/cpp>

If you create a full iterator then your structure will integrate smoothly into functions from the algorithm library. We will not need to go that far for our purpose. So we will construct a class that will be able to access nodes of our doubly linked list and move freely in either direction through the structure and, at the same time, hide the underlying pointer manipulation from the code that is outside the list class structure. In addition, this enhancement to our class structure will enable some more base C++ functionality that can be applied to our list class. Specifically, range-based loops can be used to traverse it. Take the finished version of the doubly linked list class from the previous exercise and add in the following.

1. As we saw with the STL list class, there is an extra node that is at the end of the list and is pointed to by an end pointer that we can get (in iterator form) with the `end()` command. This object is like a sentinel value that designates the end of the list and although it is in the list it is really not holding data we put in the list. You will see this type of thing again if you continue the study of data structures. Next semester in COSC 320 you will look at, and hopefully create, a tree structure called a red-black tree. These trees are used “under the hood” in the STL for objects like the map and multimap. In the red-black tree, there is a node called NIL that serves a similar purpose as our sentinel list node except that there are many nodes that point to it.

Add in another `ListNode` pointer called `endnode` alongside our head and tail pointers. The `DLinkedList` constructors will create a new `ListNode` and point `endnode` to it. So now `endnode` will be the last node in the list but it is not a data holding node. Head still points to the first node and tail still points to the last data holding node, which is the node that is pointing to the `endnode`. When the list is empty, the `endnode` still exists, the head is pointing to `endnode` and tail is `nullptr`.

2. Now we will discuss the specification for the iterator class. First we want to be able to instantiate this class from outside the linked list, say in the main or other class, with a syntax like that of the STL, for example,

```
DLinkedList<int>::iterator lstIter
```

To do that we need to put the specification for the `iterator` in the public section of the `DLinkedList` class. So `iterator` is simply an inner class and its name is just `iterator`.

The iterator is an object that can move inside the list hence it needs to be able to point to the nodes in the list. So the iterator needs to have as a data member a pointer to a `ListNode` object, in fact this is the only data member it needs. As with the linked list, we do not want this pointer to be visible outside the class since we want to control it. Hence this pointer must be in the private section of the iterator specification. Putting it in a protected section would be fine too but since we will probably not use inheritance

of this structure a private section is fine. We do not want it to be in the public section or else it will be visible outside the list class.

3. One thing that may come as a surprise is that even though the iterator is an inner class of the linked list, the linked list cannot see private members of the iterator. So although the iterator is contained in the linked list class it is still somewhat separate from the linked list class. Not all languages are like this but the developers of C++ have their reasons. You will probably need, or at least it will be convenient for you, to access this private pointer inside the Linked list class but of course not outside. To do this you can friend the link list class to the iterator by adding the following line in the specification of the iterator.

```
friend class DLinkedList;
```

4. At this point we can add in the iterator functionality, we will start with the specification.
 - (a) Create a constructor that brings in a ListNode pointer, defaulted to nullptr, that will just store the parameter pointer contents (address) into the pointer member variable you created.
 - (b) Although we are dealing with pointers in this class we are not going to be constructing any new nodes, we are just pointing to them. So there is no reason to overload the assignment or to create a copy constructor, the default ones from C++ will do the trick.
 - (c) We will need to overload operators for moving the iterator around the linked list, that is, the ones that we normally use with pointers. Overload the pre and post ++, pre and post --, += with integer right operand, -= with integer right operand, + with integer right operand, and - with integer right operand. Each of these should return an iterator object of the result of the operation. This will allow for syntax like, `lstIter = lstIter2 - 1;` and `lstIter = lstIter2 += 1;`. Overload the == and != to return a boolean that compares the pointers, if they are pointing to the same ListNode in the list then they are equivalent. Two more operators are needed for an overload, the two de-referencing operators * and ->. These will both return a reference to the value stored in the ListNode that is being pointed to by the iterator.
5. For the implementation, we are going to remove all inline implementations of code. In the previous versions of the linked list we had only one function, the ListNode constructor, that was inlined. We are going to make this implementation not inline and do all the iterator implementations after the list specification, as we usually do.
 - (a) First we will move the ListNode constructor outside the specification. I will be nice and actually give this to you so you can see the structure of implementing an inner class function outside the containing classes specification. First, in the specification of the ListNode inner class our function prototype is simply,

```
ListNode(T nodeValue);
```

The implementation of this function, which is sitting below the end of the list specification, is

```
template <class T> DLinkedList<T>::ListNode::ListNode(T nodeValue) {
    value = nodeValue;
    next = prev = nullptr;
}
```

So the general structure is the template, `DLinkedList<T>` then scope into the `ListNode` class, then scope to the function `ListNode(T nodeValue)`. Finally write the block of code that defines the function. Note that if the function has a return type then it goes between the template and `DLinkedList<T>`. You will follow the same style when implementing the iterator functions.

- (b) Now implement the functionality for each of the iterator function and overloaded operator. The header syntax follows the same style as in the `ListNode` example above. So for the logical operators the header will look like

```
template <class T>
bool DLinkedList<T>::iterator::operator==(const iterator &right)
```

as you would expect. Of course, you would follow this with a block of code that implements `==`. There is a small exception when returning an inner class type, as with all the arithmetic operator overloads. In these cases you are returning an iterator, which is an inner class. The compiler needs a little help with this because it will not see the return type of `DLinkedList<T>::iterator` as a data type. As I said in class, the reserved word `typename` is not just an alternative to the word `class` in the templating syntax, it is really more general and tells the compiler that what follows is a data type. So your header syntax for these will look like the following.

```
template <class T>
typename DLinkedList<T>::iterator DLinkedList<T>::iterator::operator++()
```

Note that we are simply putting `typename` before the `DLinkedList<T>::iterator` return type. The scoping chain down to the operator is as usual.

6. Now that we have a functional iterator we can add more functionality to the linked list class that incorporates iterators.
 - (a) Add in a `find` function that takes in a single parameter of templated type and returns an iterator to the position of the first node in the list that matches the input value. If the value is not found in the list then the iterator should be pointing to the end node.
 - (b) Add in a `begin` function that returns an iterator to the first node of the list. If the list is empty then this iterator will be pointing to the end node.
 - (c) Add in an `end` function that returns an iterator to the end node of the list.

- (d) Add in an insert function that takes parameters of an iterator and a templated value. The function is to insert the value in a new ListNode and place that ListNode in the list at the position directly before the place where the iterator is pointing. As a consequence, if the iterator is at the first node then the new node is inserted at the front of the list and if the iterator is pointing to the end node then the new node is appended to the end of the list. Don't forget the tail is pointing to the last true data node and may need to be adjusted in this operation.
- (e) Add in a remove function that takes a single parameter of an iterator. The function is to delete the node that the iterator is pointing to. Obviously, you are not to delete the end node, but the others are fair game. Again do not forget about the tail.

One nifty consequence, besides allowing us to be more efficient in our algorithms, is that now range-based loops will work with our structure.

2.3.1 Test Program for Doubly Linked List with Iterator

```
#include <ctime>
#include <iostream>

#include "DLinkedList.h"

using namespace std;

template <class T> void displayHeadTail(DLinkedList<T> &);

template <class T> class Thing {
private:
    T *d;

public:
    Thing(T val = T()) {
        d = new T;
        *d = val;
    };
    ~Thing() { delete d; }
    Thing(const Thing &obj) {
        d = new T;
        *d = *(obj.d);
    };
    const Thing operator=(const Thing &right) {
        *d = *(right.d);
        return *this;
    }
    void set(T a) { *d = a; }
    T get() { return *d; }
    friend ostream &operator<<(ostream &out, Thing &t) {
        out << *t.d;
        return out;
    }
};

void div(string s = "") {
    cout << "\n" << s << "-----\n\n";
}
```

```
int main() {
    srand(time(0));
    DLinkedList<int> lst;

    div("Push back and delete tests ");

    lst.push_back(12);
    lst.push_back(4);
    lst.push_back(54);
    lst.push_back(10);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    lst.push_back(101);
    lst.push_back(17);
    lst.push_back(21);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    lst.deleteNode(12);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    lst.deleteNode(101);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    lst.deleteNode(1234);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    lst.deleteNode(21);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    lst.push_back(112);
    lst.push_back(14);
    lst.push_back(154);
    lst.push_back(110);

    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    div("Print reversed ");

    lst.displayList();
    cout << endl;
    lst.displayList(false, true);
    cout << endl;

    div("Insert Tests ");
```



```
lst.clear();
displayHeadTail(lst);

lst.clear();
lst.insertNode(45);
lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.insertNode(21);
lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.insertNode(57);
lst.displayList();
cout << endl;
displayHeadTail(lst);

for (int i = 0; i < 10; i++)
    lst.insertNode(rand() % 100);

lst.displayList();
cout << endl;
displayHeadTail(lst);

div("Push front tests ");

lst.clear();
lst.push_front(12);
lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.push_front(15);
lst.displayList();
cout << endl;
displayHeadTail(lst);

lst.push_front(25);
lst.displayList();
cout << endl;
displayHeadTail(lst);

for (int i = 0; i < 10; i++)
    lst.push_front(rand() % 100);

lst.displayList();
cout << endl;
displayHeadTail(lst);

div("Pop front and back with indexing tests ");

try {
    cout << lst.pop_front() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    cout << lst.pop_front() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);
}
```

```
    cout << lst.pop_front() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    cout << lst.pop_front() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    cout << lst.pop_back() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    cout << lst.pop_back() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

    cout << lst.pop_back() << endl;
    lst.displayList();
    cout << endl;
    displayHeadTail(lst);

} catch (string err) {
    cout << err << endl;
}

try {
    cout << lst[5] << endl;
    cout << lst[2] << endl;
    cout << lst[6] << endl;
    cout << lst[21] << endl;
} catch (string err) {
    cout << err << endl;
}

lst.displayList();
cout << endl;

try {
    lst[5] = 17;
    lst[1] = 29;
    lst[3] = 11;
    lst[52] = 7;
} catch (string err) {
    cout << err << endl;
}

lst.displayList();
cout << endl;
displayHeadTail(lst);

try {
    lst.set(5, -17);
    lst.set(1, -29);
    lst.set(3, -11);
    lst.set(52, -7);
} catch (string err) {
    cout << err << endl;
}

lst.displayList();
```

```
cout << endl;
displayHeadTail(lst);

div("CC and = tests ");

DLinkedList<int> L1;
for (int i = 0; i < 5; i++)
    L1.push_back(rand() % 10000 + 1);

DLinkedList<int> L2 = L1;
DLinkedList<int> L3(L1);

L1.displayList();
cout << endl;
L2.displayList();
cout << endl;
L3.displayList();
cout << endl << endl;

L1[3] = 1234567;
L3[3] = 7654321;

L1.displayList();
cout << endl;
L2.displayList();
cout << endl;
L3.displayList();
cout << endl << endl;

L3 = L2 = L1;

L1.displayList();
cout << endl;
L2.displayList();
cout << endl;
L3.displayList();
cout << endl << endl;

L1[3] = 987654321;
L2[1] = -25;

L1.displayList();
cout << endl;
L2.displayList();
cout << endl;
L3.displayList();
cout << endl << endl;

L1 = L1;
cout << L1 << endl;
cout << endl;

div("Reverse tests ");

L2 = L1.reverse();
cout << L1 << endl;
cout << L2 << endl;
cout << endl;

L2.pop_front();
L2.pop_back();
cout << L1 << endl;
cout << L2 << endl;
cout << endl;
```

```

div("Sort tests ");

L2 = L1.sort();
cout << L1 << endl;
cout << L2 << endl;
cout << endl;

L1 = L1.sort();
cout << L1 << endl;
cout << endl;

div("Iterator Tests ");

cout << L1 << endl;
DLinkedList<int>::iterator lstIter = L1.begin();

for (DLinkedList<int>::iterator it = L1.begin(); it != L1.end(); it++)
    cout << *it << " ";
cout << endl;

lstIter += 2;
*lstIter = 12345;

for (DLinkedList<int>::iterator it = L1.begin(); it != L1.end(); it++)
    cout << *it << " ";
cout << endl;

// Even a range-based for loop works with this structure.
for (auto element : L1)
    cout << element << " ";
cout << endl;

for (int element : L1)
    cout << element << " ";
cout << endl;

// Default CC and = work as well with the iterator, no need to overload.
lstIter = L1.end();
cout << *lstIter << endl;

lstIter = L1.begin();
lstIter += 3;
DLinkedList<int>::iterator lstIter2(lstIter);
cout << *lstIter2 << endl;
lstIter2--;
cout << *lstIter2 << endl;
lstIter2 -= 2;
cout << *lstIter2 << endl;
lstIter2 -= 5;
cout << *lstIter2 << endl;
lstIter2 += 2;
cout << *lstIter2 << endl;
lstIter2 += 200;
cout << *lstIter2 << endl;
--lstIter2;
cout << *lstIter2 << endl;
--lstIter2;
cout << *lstIter2 << endl;
++lstIter2;
cout << *lstIter2 << endl;
cout << *lstIter << endl;

lstIter = L1.begin();

```

```

lstIter2 = lstIter += 2;
cout << *lstIter2 << endl;
cout << *lstIter << endl;

lstIter2 = lstIter++;
cout << *lstIter2 << endl;
cout << *lstIter << endl;

lstIter2 = ++lstIter;
cout << *lstIter2 << endl;
cout << *lstIter << endl;

lstIter2 = lstIter--;
cout << *lstIter2 << endl;
cout << *lstIter << endl;

lstIter2 = --lstIter;
cout << *lstIter2 << endl;
cout << *lstIter << endl;

L2 = L1;

cout << L1 << endl;
cout << L2 << endl;

lstIter = L1.begin();
lstIter2 = L2.begin();

lstIter += 2;
lstIter2 += 3;
*lstIter = 23;
*lstIter2 = 42;

cout << L1 << endl;
cout << L2 << endl;

if (L1.find(23) != L1.end()) {
    cout << "23 found" << endl;
} else {
    cout << "23 not found" << endl;
}

if (L1.find(123) != L1.end()) {
    cout << "123 found" << endl;
} else {
    cout << "123 not found" << endl;
}

lstIter = L1.find(23);
cout << *lstIter << endl;
lstIter++;
cout << *lstIter << endl;

lstIter = L1.end() - 1;
cout << *lstIter << endl;
lstIter = L1.begin() + 2;
cout << *lstIter << endl;

DLinkedList<int *> L4;

for (int i = 0; i < 10; i++) {
    int *nint = new int;
    *nint = rand() % 100;
    L4.push_back(nint);
}

```

```

}

for (DLinkedList<int *>::iterator it = L4.begin(); it != L4.end(); it++)
    cout << **it << " ";
cout << endl;

// Remove the new ints we created.
for (DLinkedList<int *>::iterator it = L4.begin(); it != L4.end(); it++)
    delete *it;

div("Iterator Insert Tests ");
cout << L1 << endl;
lstIter = L1.begin() + 3;
L1.insert(lstIter, 12345);
cout << L1 << endl;
displayHeadTail(L1);
L1.insert(L1.begin(), 54321);
cout << L1 << endl;
displayHeadTail(L1);
L1.insert(L1.end(), 77777);
cout << L1 << endl;
displayHeadTail(L1);

div("Iterator Remove Tests ");
cout << L1 << endl;
lstIter = L1.begin() + 2;
L1.remove(lstIter);
cout << L1 << endl;
displayHeadTail(L1);
L1.remove(L1.begin());
cout << L1 << endl;
displayHeadTail(L1);
L1.remove(L1.end());
cout << L1 << endl;
displayHeadTail(L1);
L1.remove(L1.end() - 1);
cout << L1 << endl;
displayHeadTail(L1);

cout << L1.isEmpty() << endl;

while (!L1.isEmpty()) {
    L1.remove(L1.begin());
    cout << L1 << endl;
    displayHeadTail(L1);
}
cout << L1.isEmpty() << endl;

// Go too far.
L1.remove(L1.begin());
cout << L1 << endl;
cout << L1.isEmpty() << endl;

DLinkedList<Thing<double>> Things;

for (int i = 0; i < 10; i++) {
    double t = 1.0 * rand() / RAND_MAX;
    Thing<double> thing(t);
    Things.push_back(thing);
}

DLinkedList<Thing<double>>::iterator thingit;
for (DLinkedList<Thing<double>>::iterator it = Things.begin();
     it != Things.end(); it++)

```

```

    cout << *it << " ";
    cout << endl;

    DLinkedList<Thing<double>*> PThings;
    DLinkedList<Thing<double>*>::iterator pthingit;
    for (int i = 0; i < 15; i++) {
        double t = 1.0 * rand() / RAND_MAX;
        Thing<double> *thing = new Thing<double>;
        thing->set(t);
        PThings.push_back(thing);
    }

    for (DLinkedList<Thing<double>*>::iterator it = PThings.begin();
         it != PThings.end(); it++)
        cout << it->get() << " ";
    cout << endl;

    for (auto element : PThings)
        cout << element->get() << " ";
    cout << endl;

    for (auto element : PThings)
        element->set(element->get() + 1);

    for (auto element : PThings)
        cout << element->get() << " ";
    cout << endl;

    // Delete the things created for the list.
    for (auto element : PThings)
        delete element;

    return 0;
}

template <class T> void displayHeadTail(DLinkedList<T> &L) {
    cout << "Head: ";
    try {
        cout << L.peekHead() << endl;
    } catch (string s) {
        cout << s << endl;
    }

    cout << "Tail: ";
    try {
        cout << L.peekTail() << endl;
    } catch (string s) {
        cout << s << endl;
    }
}

```

2.3.2 Program Output

```

Push back and delete tests -----

12 4 54 10
Head: 12
Tail: 10
12 4 54 10 101 17 21
Head: 12
Tail: 21
4 54 10 101 17 21
Head: 4
Tail: 21

```

4 54 10 17 21

Head: 4

Tail: 21

4 54 10 17 21

Head: 4

Tail: 21

4 54 10 17

Head: 4

Tail: 17

4 54 10 17 112 14 154 110

Head: 4

Tail: 110

Print reversed -----

4 54 10 17 112 14 154 110

110 154 14 112 17 10 54 4

Insert Tests -----

Head: Null pointer exception.

Tail: Null pointer exception.

45

Head: 45

Tail: 45

21 45

Head: 21

Tail: 45

21 45 57

Head: 21

Tail: 57

17 21 27 45 55 57 66 70 78 85 88 92 96

Head: 17

Tail: 96

Push front tests -----

12

Head: 12

Tail: 12

15 12

Head: 15

Tail: 12

25 15 12

Head: 25

Tail: 12

71 94 14 36 23 92 98 10 83 1 25 15 12

Head: 71

Tail: 12

Pop front and back with indexing tests -----

71

94 14 36 23 92 98 10 83 1 25 15 12

Head: 94

Tail: 12

94

14 36 23 92 98 10 83 1 25 15 12

Head: 14

Tail: 12

14

36 23 92 98 10 83 1 25 15 12

Head: 36

Tail: 12


```
36
23 92 98 10 83 1 25 15 12
Head: 23
Tail: 12
12
23 92 98 10 83 1 25 15
Head: 23
Tail: 15
15
23 92 98 10 83 1 25
Head: 23
Tail: 25
25
23 92 98 10 83 1
Head: 23
Tail: 1
1
98
Index Out of Bounds Exception
23 92 98 10 83 1
Index Out of Bounds Exception
23 29 98 11 83 17
Head: 23
Tail: 17
Index Out of Bounds Exception
23 -29 98 -11 83 -17
Head: 23
Tail: -17

CC and = tests -----

1492 2788 5785 9531 5209
1492 2788 5785 9531 5209
1492 2788 5785 9531 5209

1492 2788 5785 1234567 5209
1492 2788 5785 9531 5209
1492 2788 5785 7654321 5209

1492 2788 5785 1234567 5209
1492 2788 5785 1234567 5209
1492 2788 5785 1234567 5209

1492 2788 5785 987654321 5209
1492 -25 5785 1234567 5209
1492 2788 5785 1234567 5209

[1492, 2788, 5785, 987654321, 5209]

Reverse tests -----

[1492, 2788, 5785, 987654321, 5209]
[5209, 987654321, 5785, 2788, 1492]

[1492, 2788, 5785, 987654321, 5209]
[987654321, 5785, 2788]

Sort tests -----

[1492, 2788, 5785, 987654321, 5209]
[1492, 2788, 5209, 5785, 987654321]
```

```
[1492, 2788, 5209, 5785, 987654321]
```

```
Iterator Tests -----
```

```
[1492, 2788, 5209, 5785, 987654321]
1492 2788 5209 5785 987654321
1492 2788 12345 5785 987654321
1492 2788 12345 5785 987654321
1492 2788 12345 5785 987654321
0
5785
12345
1492
1492
12345
0
987654321
5785
987654321
5785
12345
12345
12345
5785
987654321
987654321
987654321
5785
12345
12345
[1492, 2788, 12345, 5785, 987654321]
[1492, 2788, 12345, 5785, 987654321]
[1492, 2788, 23, 5785, 987654321]
[1492, 2788, 12345, 42, 987654321]
23 found
123 not found
23
5785
987654321
23
22 49 8 77 76 52 56 46 30 52
```

```
Iterator Insert Tests -----
```

```
[1492, 2788, 23, 5785, 987654321]
[1492, 2788, 23, 12345, 5785, 987654321]
Head: 1492
Tail: 987654321
[54321, 1492, 2788, 23, 12345, 5785, 987654321]
Head: 54321
Tail: 987654321
[54321, 1492, 2788, 23, 12345, 5785, 987654321, 77777]
Head: 54321
Tail: 77777
```

```
Iterator Remove Tests -----
```

```
[54321, 1492, 2788, 23, 12345, 5785, 987654321, 77777]
[54321, 1492, 23, 12345, 5785, 987654321, 77777]
Head: 54321
Tail: 77777
[1492, 23, 12345, 5785, 987654321, 77777]
Head: 1492
```

```

Tail: 77777
[1492, 23, 12345, 5785, 987654321, 77777]
Head: 1492
Tail: 77777
[1492, 23, 12345, 5785, 987654321]
Head: 1492
Tail: 987654321
0
[23, 12345, 5785, 987654321]
Head: 23
Tail: 987654321
[12345, 5785, 987654321]
Head: 12345
Tail: 987654321
[5785, 987654321]
Head: 5785
Tail: 987654321
[987654321]
Head: 987654321
Tail: 987654321
[]
Head: Null pointer exception.
Tail: Null pointer exception.
1
[]
1
0.924959 0.30594 0.466223 0.889388 0.585376 0.4654 0.52507 0.352107 0.177954 0.662577
0.778294 0.0873852 0.353592 0.180575 0.370474 0.872852 0.925937 0.128912 0.66548 0.0576207 0.252175 0
0.778294 0.0873852 0.353592 0.180575 0.370474 0.872852 0.925937 0.128912 0.66548 0.0576207 0.252175 0
1.77829 1.08739 1.35359 1.18058 1.37047 1.87285 1.92594 1.12891 1.66548 1.05762 1.25218 1.10401 1.35359

Push back and delete tests -----

12 4 54 10
Head: 12
Tail: 10
12 4 54 10 101 17 21
Head: 12
Tail: 21
4 54 10 101 17 21
Head: 4
Tail: 21
4 54 10 17 21
Head: 4
Tail: 21
4 54 10 17 21
Head: 4
Tail: 21
4 54 10 17
Head: 4
Tail: 17
4 54 10 17 112 14 154 110
Head: 4
Tail: 110

Print reversed -----

4 54 10 17 112 14 154 110
110 154 14 112 17 10 54 4

Insert Tests -----

Head: Null pointer exception.
Tail: Null pointer exception.

```

```
45
Head: 45
Tail: 45
21 45
Head: 21
Tail: 45
21 45 57
Head: 21
Tail: 57
4 5 16 21 21 29 45 57 69 77 77 85 90
Head: 4
Tail: 90
```

Push front tests -----

```
12
Head: 12
Tail: 12
15 12
Head: 15
Tail: 12
25 15 12
Head: 25
Tail: 12
3 75 78 90 36 49 81 46 67 22 25 15 12
Head: 3
Tail: 12
```

Pop front and back with indexing tests -----

```
3
75 78 90 36 49 81 46 67 22 25 15 12
Head: 75
Tail: 12
75
78 90 36 49 81 46 67 22 25 15 12
Head: 78
Tail: 12
78
90 36 49 81 46 67 22 25 15 12
Head: 90
Tail: 12
90
36 49 81 46 67 22 25 15 12
Head: 36
Tail: 12
12
36 49 81 46 67 22 25 15
Head: 36
Tail: 15
15
36 49 81 46 67 22 25
Head: 36
Tail: 25
25
36 49 81 46 67 22
Head: 36
Tail: 22
22
81
Index Out of Bounds Exception
36 49 81 46 67 22
Index Out of Bounds Exception
36 29 81 11 67 17
```

```

Head: 36
Tail: 17
Index Out of Bounds Exception
36 -29 81 -11 67 -17
Head: 36
Tail: -17

```

```
CC and = tests -----
```

```

2340 8560 5916 9362 586
2340 8560 5916 9362 586
2340 8560 5916 9362 586

```

```

2340 8560 5916 1234567 586
2340 8560 5916 9362 586
2340 8560 5916 7654321 586

```

```

2340 8560 5916 1234567 586
2340 8560 5916 1234567 586
2340 8560 5916 1234567 586

```

```

2340 8560 5916 987654321 586
2340 -25 5916 1234567 586
2340 8560 5916 1234567 586

```

```
[2340, 8560, 5916, 987654321, 586]
```

```
Reverse tests -----
```

```

[2340, 8560, 5916, 987654321, 586]
[586, 987654321, 5916, 8560, 2340]

```

```

[2340, 8560, 5916, 987654321, 586]
[987654321, 5916, 8560]

```

```
Sort tests -----
```

```

[2340, 8560, 5916, 987654321, 586]
[586, 2340, 5916, 8560, 987654321]

```

```
[586, 2340, 5916, 8560, 987654321]
```

```
Iterator Tests -----
```

```

[586, 2340, 5916, 8560, 987654321]
586 2340 5916 8560 987654321
586 2340 12345 8560 987654321
586 2340 12345 8560 987654321
586 2340 12345 8560 987654321
0
8560
12345
586
586
12345
0
987654321
8560
987654321
8560
12345

```

```
12345
12345
8560
987654321
987654321
987654321
8560
12345
12345
[586, 2340, 12345, 8560, 987654321]
[586, 2340, 12345, 8560, 987654321]
[586, 2340, 23, 8560, 987654321]
[586, 2340, 12345, 42, 987654321]
23 found
123 not found
23
8560
987654321
23
7 92 29 6 10 78 11 47 7 32
```

Iterator Insert Tests -----

```
[586, 2340, 23, 8560, 987654321]
[586, 2340, 23, 12345, 8560, 987654321]
Head: 586
Tail: 987654321
[54321, 586, 2340, 23, 12345, 8560, 987654321]
Head: 54321
Tail: 987654321
[54321, 586, 2340, 23, 12345, 8560, 987654321, 77777]
Head: 54321
Tail: 77777
```

Iterator Remove Tests -----

```
[54321, 586, 2340, 23, 12345, 8560, 987654321, 77777]
[54321, 586, 23, 12345, 8560, 987654321, 77777]
Head: 54321
Tail: 77777
[586, 23, 12345, 8560, 987654321, 77777]
Head: 586
Tail: 77777
[586, 23, 12345, 8560, 987654321, 77777]
Head: 586
Tail: 77777
[586, 23, 12345, 8560, 987654321]
Head: 586
Tail: 987654321
0
[23, 12345, 8560, 987654321]
Head: 23
Tail: 987654321
[12345, 8560, 987654321]
Head: 12345
Tail: 987654321
[8560, 987654321]
Head: 8560
Tail: 987654321
[987654321]
Head: 987654321
Tail: 987654321
[]
Head: Null pointer exception.
```

Tail: Null pointer exception.

```
1
[]
1
0.243001 0.936229 0.290208 0.336004 0.515189 0.551175 0.336719 0.589724 0.798326 0.641761
0.302737 0.215069 0.200152 0.0540039 0.0848019 0.407794 0.707379 0.319467 0.647166 0.17161
0.729682 0.943846 0.906085 0.879051 0.0825647
0.302737 0.215069 0.200152 0.0540039 0.0848019 0.407794 0.707379 0.319467 0.647166 0.17161
0.729682 0.943846 0.906085 0.879051 0.0825647
1.30274 1.21507 1.20015 1.054 1.0848 1.40779 1.70738 1.31947 1.64717 1.17161 1.72968 1.94385
1.90608 1.87905 1.08256
```

3 Algorithm and Structure Analysis

This part is going to test the speed of some standard operations between our different structures of our single and double linked list, as well as the STL vector and STL list.

3.1 Timing Processes in General

There are many ways to time a process inside your code. We will use the chrono library that is built into C++. If you have never used this library here is how you set it up.

First you include the chrono library with your other includes.

```
#include <chrono>
```

Second, you don't need to include the sub-namespace but it makes the code a bit easier. So we will, below the std namespace, add in,

```
using namespace std::chrono;
```

Now, inside your code, you can time a process using the following syntax.

```
auto start = high_resolution_clock::now();
// Process to time.
auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start);
cout << "Time to append with linked list: " << duration.count() / 1000000.0
    << " seconds" << endl;
```

where you replace `// Process to time.` with a block of code or a function call to a function you wish to time. So for example, if we wanted to time the bubble sort on an array `A` we could use the block of code.

```
auto start = high_resolution_clock::now();
bubbleSort(A, size);
auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start);
cout << "Time to sort array with bubble sort: "
    << duration.count() / 1000000.0 << " seconds" << endl;
```

Remember that `auto` is a declaration and then the variable has that type from here on out, so if you reuse `start`, `stop`, and `duration` to time another process you will remove the `auto` declarations. For example,

```
start = high_resolution_clock::now();
selectionSort(A, size);
stop = high_resolution_clock::now();
duration = duration_cast<microseconds>(stop - start);
cout << "Time to sort array with bubble sort: "
      << duration.count() / 1000000.0 << " seconds" << endl;
```

A note on this, notice that the duration will come out in microseconds, one one millionth of a second. You can also use nanoseconds if you would like, one one billionth of a second. Just change all the microseconds to nanoseconds and instead of dividing by 1000000.0 you divide by 1000000000.0. If you do change to nanoseconds use nanoseconds the whole way through, do not mix them since duration will be defined specifically to be one of these.

One very important aspect of using empirical timings to analyze algorithm efficiency is that you do all your timings on the same machine. Doing some timings on one computer and then doing other timings on another computer cannot be compared due to the differences in hardware, operating systems, background processes, etc. Also, virtual machines add a layer of operation, so I would do all my timings in the Linux lab unless you installed Linux directly on the metal. Another thing you will want to do is to shut down extra applications you do not need. Such as word processors, spreadsheets, browsers, games, etc. so that their operation does not interfere with the running of your program.

Another consideration along these lines is that during a timing run your computer could do a background process that is computationally intensive or resource intensive, such as, accessing the hard drive, that may skew your timing results. You will want to run your timing tests several times to see what times are consistent and use those times. Better yet is to take several times, say 5 runs, and average the times together. You don't need to do that here but at least do several runs and discard times that do not fit with the others.

3.2 Timing Insertions

The first thing we are going to time is the speed of loading large amounts of data into some of the standard data structures we have been working with and creating.

Whenever you learn about a new data structure, such as the linked list, you should test it out against the other data structures you already know, like the STL array, list, and vector, to see where it is more efficient and where it is less efficient. There are pros and cons to every data structure. They may be more efficient in one manner but less efficient in another.

This exercise will test the speed of appending an element to the front and back of several data structures.

1. In the main,
 - Include the following: `iostream`, `list`, `vector`, `ctime`, `cstdlib`, `chrono`, and (obviously) the header file for the two linked list classes you created in the last part, that is, your singly and doubly linked list, and the linked list class that is in the example code (the ones we went over in lecture). If you did the extra credit

exercise you can use either for the doubly linked list, you do not need to include both.

- Have the program ask the user for the number of items to insert into the structures.
- Set up timing blocks to time that number of front insertions of random integers into your linked list, your doubly linked list, the textbook linked list, the STL vector, and the STL list. Report all these times in seconds. Obviously, all of the templated types will be `int`.

We need to do two tricks for this timing. Since the textbook linked list does not have a push front command nor does the vector. For the textbook linked list we can simply insert smaller numbers, these will always be pushed onto the front. For example, we could simply time,

```
for (int i = 0; i < trials; i++)
    textlst.insertNode(trials - i);
```

For the vector, we can use its iterator. For example,

```
for (int i = 0; i < trials; i++)
    vec.insert(vec.begin(), rand());
```

- Clear each of the structures outside the timing loops to get them ready for the next set of timings. You may need to write a clear function for the textbook linked list, it is simply the same code as the destructor but make sure you set the head to `nullptr` at the end.
 - Set up timing blocks to time that number of back insertions of random integers into all five structures. Report all these times in seconds.
2. Run the program on data sizes of 10,000, 25,000, 50,000, 75,000, 100,000, 150,000, 200,000 items. An example of the output on my machine for 20,000 items is below.

```
Input the number of values to store: 20000
```

```
-----
```

```
Time to push front with linked list: 0.003201 seconds
Time to push front with double linked list: 0.001403 seconds
Time to push front with textbook linked list: 0.000766 seconds
Time to insert at front with vector: 0.01481 seconds
Time to push front with STL linked list: 0.002042 seconds
```

```
-----
```

```
Time to push back with linked list: 0.000539 seconds
Time to push back with double linked list: 0.000556 seconds
Time to push back with textbook linked list: 0.729477 seconds
Time to push back with vector: 0.000587 seconds
Time to push back with STL list: 0.001672 seconds
```

3. Using a spreadsheet, such as LibreOffice Calc or Excel, plot the data. You will want to use an XY scatter chart with the sizes on the *x*-axis and times on the *y*-axis. Put all five lines on the same graph, that is, all three linked lists, SLT list, and STL vector.

4. In a word processor, you can use LibreOffice Writer, Word, or any word processor that will allow you to export the document to a PDF file. LibreOffice Writer has a toolbar button for this. In the document include tables of all the data you acquired from the runs you did above, the charts you created from the data and the answers, in complete sentences, to the following.
 - How do the times to insert at the front of each structure compare? Are they consistent? Do the curves have the same shape (possibly indicating that they have the same computational cost). Which structures perform about the same and which are different?
 - How do the times to insert at the end of each structure compare? Are they consistent? Do the curves have the same shape (possibly indicating that they have the same computational cost). Which structures perform about the same and which are different?
 - What part of the algorithms or the design of the data structures contribute to the timing differences?
 - Export the document to a PDF file and include it with your code in the zip file you upload.

3.3 Timing Sorts

This exercise will test the speed of the linked list insert function verses sorting an array of the same elements with the Bubble Sort, Selection Sort, and the Insertion Sort. Recall that we created the insert function for the linked list to add in the elements in order. So using the insert instead of append or pushback will result in a sorted list (from smallest to largest).

1. Copy over the two linked list classes you created in the first two exercises. Also include these in the main.
2. Take the Bubble Sort, Selection Sort, and the Insertion Sort code below and template the functions to work with any data type with valid relational operators. The ones below are written for arrays of integers, so be careful on which variables need to be changed to T and which stay as integers.

```
void bubbleSort(int array[], int size) {
    int maxElement;
    int index;

    for (maxElement = size - 1; maxElement > 0; maxElement--)
        for (index = 0; index < maxElement; index++)
            if (array[index] > array[index + 1])
                swap(array[index], array[index + 1]);
}

void selectionSort(int array[], int size) {
    int minIndex, minValue;

    for (int start = 0; start < (size - 1); start++) {
```

```

        minIndex = start;
        minValue = array[start];
        for (int index = start + 1; index < size; index++) {
            if (array[index] < minValue) {
                minValue = array[index];
                minIndex = index;
            }
        }
        swap(array[minIndex], array[start]);
    }
}

void insertionSort(int array[], int size) {
    for (int itemsSorted = 1; itemsSorted < size; itemsSorted++) {
        int temp = array[itemsSorted];
        int loc = itemsSorted - 1;
        while (loc >= 0 && array[loc] > temp) {
            array[loc + 1] = array[loc];
            loc = loc - 1;
        }
        array[loc + 1] = temp;
    }
}

```

3. In the main, set the seed of the random number generator to the clock as usual.
4. Ask the user how many data items they wish to store.
5. Create empty linked lists of integers, an empty vector of integers, and a dynamic array of integers of the size to store the data items.
6. Using the vector, populate the vector with the number of data items the user requested each of which is a random integer. This is simply storing the random values so that we can load them into each structure and hence time the sorting on the same data set.
7. Separately, time the process of inserting the data from the vector into the two linked list structures. That is, take each element of the vector one at a time and use the list insert function to insert it into the linked list, effectively, sorting the data.
8. Time the process of inserting the data from the vector into the array and then sorting the array with the Bubble Sort.
9. Time the process of inserting the data from the vector into the array and then sorting the array with the Insertion Sort.
10. Time the process of inserting the data from the vector into the array and then sorting the array with the Selection Sort.
11. Run the program on sizes of 10,000, 25,000, 50,000, 75,000, and 100,000 items.

An example run on my laptop with 20,000 data items is below.

```

Input the number of values to store: 20000
Time to sort with linked list: 0.892987 seconds
Time to sort with double linked list: 0.861348 seconds
Time to sort array with bubble sort: 1.81093 seconds
Time to sort array with insertion sort: 0.251529 seconds
Time to sort array with selection sort: 0.447638 seconds

```

12. Using a spreadsheet, such as LibreOffice Calc or Excel, plot the sort times verses the number of data items. Put all five lines on the same chart. You will want to use an XY scatter chart with the sizes on the x -axis and times on the y -axis.
13. In a word processor, you can use LibreOffice Writer, Word, or any word processor that will allow you to export the document to a PDF file. LibreOffice Writer has a toolbar button for this. In the document include tables of all the data you acquired from the runs you did above, the charts you created from the data and the answers, in complete sentences, to the following.
 - How do the times for these three methods compare?
 - Do the curves have approximately the same shape? That is, do they increase along a parabolic-like curve, are they both straight lines, do they have the same types of curves in about the same places?
 - What do you think is accounting for the differences?

Export the document to a PDF file and include it with your code in the zip file you upload.

4 Application: Scalable Vector Graphics (SVG) Files

This exercise is going to combine our work with the inheritance hierarchy we created with the shapes examples and the doubly linked list to store the data. At the end we will export the data to a scalable vector graphics (SVG) file that will allow us to view the objects in a graphics viewer or web browser.

You will be creating all structures used in this program yourself. No use of STL or algorithm library functions is permitted.

4.1 Shape Inheritance Hierarchy Description

1. First we will update the shape classes inheritance hierarchy. The base class will still be the Shape class. Its data members are going to change to the following,
 - bordercolor which is a string.
 - fillcolor which is a string.
 - borderwidth which is an integer.
 - opacity which is a double.
2. The shape class should have sets and gets for all the data members. The sets should, of course, do validity checking.

The strings bordercolor and fillcolor must be from the following list, any other strings are invalid values.

black	darkgreen	lightgray	red
blue	darkorange	lightgreen	violet
brown	darkred	magenta	yellow
cyan	gold	maroon	transparent
darkblue	gray	navy	
darkcyan	green	orange	
darkgray	lightcyan	purple	

Similarly, `borderwidth` should be at least 1 and `opacity` is to be between 0 and 1 inclusive.

3. `Shape` needs three more functions all of which are to be dynamically bound for polymorphism.
 - A function `getBounds` that returns a pointer to an integer array which is purely virtual.
 - A void `draw` function.
 - A `svgcode` function that returns a string.

We will discuss what these functions do at the end after we discuss the child classes.

4. A `Rectangle` class inherited off of `Shape`. The `Rectangle` class will have four integer data members: `tlx`, `tly`, `width`, and `height`. The meaning of these values will become evident as we go along but for now, the `tlx` and `tly` values are the (x, y) coordinates for the top left vertex of the rectangle and the `width` and `height` are the width and height of the rectangle. One thing to note is that we are using integer positions here because they are going to correspond to pixel positions on an image.

Create getters and setters for each data member, as usual, validity check the setters. The `width` and `height` must both be at least 1, but the `tlx` and `tly` do not have any bounds stipulations.

5. A `Square` class inherited off of `Rectangle`. The `square` class does not need any additional data or accessor/mutator functions.
6. A `Circle` class inherited off of `Shape`. The `circle` class will have three integer data members, `cx`, `cy`, and `radius`. Again, the data will be made clearer as we go along. For now, the `cx` and `cy` are the (x, y) coordinates of the center of the circle and `radius` is the radius of the circle.

Create accessors and mutators for each of the data items. The `radius` must be at least 1 but there are no restrictions on `cx` and `cy`.

7. A `Triangle` class inherited off of `Shape`. The `triangle` class has one data member, a pointer to an integer array. This array will store the 6 data items for the three vertices of the triangle. So if the three vertices to the triangle are (x_1, y_1) , (x_2, y_2) , and (x_3, y_3) , the array will have the data $x_1, y_1, x_2, y_2, x_3, y_3$ in it in that order. Clearly we can get

by on this with a stack memory array but I want this to be dynamic. Again, you want accessors and mutators for this array, there are no bounds stipulations on the x and y coordinate values.

4.2 User Interface Specifications

Before we continue with the details lets skip up to the main and describe the program goal here. When the application is run the user will be presented with the following menu.

```
1. Add Rectangle
2. Add Square
3. Add Triangle
4. Add Circle
5. Add Random Rectangles
6. Add Random Squares
7. Add Random Triangles
8. Add Random Circles
9. Add Random Objects
10. View Object Database
11. View Object Database in SVG Format
12. Save SVG File
13. Clear Shape Database
14. Quit
Choice:
```

The first four options will ask the user for all the needed data to create one copy of the intended object. The next four options will ask the user how many of those objects to create and then create that many random desired objects. Option 9 will ask the user the number of objects to create and it will create that many object selecting at random which of the four objects to create. So if the user inputs 10 then that option could create 3 rectangles, one square, 2 triangles, and 4 circles.

Option 10 prints to the console a list of all objects in the database with their attributes in the following format. The first is a rectangle, second is a square, third is a triangle, and last is the circle. This option simply prints out the size of the linked list (i.e. size of the database) and then calls the draw function on each element in the database, polymorphism takes care of the rest. Note that the border, fill, border width, and opacity are all stored in the Shape base class, so use its draw function to print out this portion of the output, no reason to redo the same code in the derived classes.

Number of Objects in Database = 4

```
Rectangle Anchor Point: (-547, -989) Width: 608 Height: 652 Border: lightgray
Fill: yellow Border Width: 4 Opacity: 0.587987
```

```
Rectangle Anchor Point: (37, -72) Width: 311 Height: 311 Border: yellow
Fill: darkorange Border Width: 1 Opacity: 0.262964
```

```
Triangle Points: (-247, -347) (-370, 944) (-727, -701) Border: yellow
Fill: transparent Border Width: 5 Opacity: 0.290014
```

```
Circle Center: (334, 310) Radius: 538 Border: lightgray Fill: black
Border Width: 1 Opacity: 0.897517
```

Option 11 prints out the database but in SVG format for each object. We will discuss the svg format in more detail later. For these objects the svg format is,

Number of Objects in Database = 4

```
<rect x="-547" y="-989" width="608" height="652" stroke="lightgray" fill="yellow"
stroke-width="4" opacity="0.587987" />

<rect x="37" y="-72" width="311" height="311" stroke="yellow" fill="darkorange"
stroke-width="1" opacity="0.262964" />

<polyline points="-247, -347 -370, 944 -727, -701 -247, -347" stroke="yellow"
fill="transparent" stroke-width="5" opacity="0.290014" />

<circle cx="334" cy="310" r="538" stroke="lightgray" fill="black" stroke-width="1"
opacity="0.897517" />
```

This option simply prints out the size of the linked list (i.e. size of the database) and then calls the `svgcode` function on each element in the database, polymorphism takes care of the rest. Note that the border, fill, border width, and opacity are all stored in the `Shape` base class, so use its `svgcode` function to construct that portion of the string, no reason to redo the same code in the derived classes. These are actually the text strings that will be saved to the svg file to create the image. Again more on that later.

Option 12 asks the user for a filename. The program will then save the database contents to an svg file that can be viewed with some graphics viewers and most web browsers, such as firefox. If the user types in a filename that does not have a “.svg” extension then the program should add it on automatically. That is, if the user types in `image1` the file will be `image1.svg`.

Option 13 will clear the contents of the entire database. Option 14 will end the program.

Obviously we could make this far more user friendly by allowing the user to remove portions of the database and edit entries that have been already entered. While this would be nice to have it is not necessary here to get a feel for how the storage and inheritance structures are working together. If this were a course in GUI design then these options, along with several others, would be added.

4.2.1 User Interface Specifications Details

- Option #1: Add a single Rectangle. When the user selects to add a rectangle the following sequence of inputs will come up. The program will ask for the x and y coordinates of the upper left vertex to the rectangle (called the anchor point), then the width and height of the rectangle. It will then print out a list of the colors we are using (svg actually supports far more but 25 is enough), the user then selects the border color from the list. The list is displayed again and the user selects the fill color from the list. Finally, the program asks for the border width and the opacity level. The range of acceptable values is given on the input line. If the user inputs something outside that range the program will ask for the input again. Furthermore, the program will also detect if the data type of the input is correct, so if the user inputs a string

where they were supposed to input a number the program will not crash but instead ask for a valid input. Again, more on that later.

```
Choice: 1
Input Upper Left X [-1000, 1000]: -102
Input Upper Left Y [-1000, 1000]: 123
Input Width [1, 1000]: 401
Input Height [1, 1000]: 500

1: black      2: blue      3: brown      4: cyan      5: darkblue
6: darkcyan   7: darkgray   8: darkgreen  9: darkorange 10: darkred
11: gold      12: gray      13: green     14: lightcyan 15: lightgray
16: lightgreen 17: magenta   18: maroon    19: navy      20: orange
21: purple    22: red       23: violet    24: yellow    25: transparent
Select Border Color: 4

1: black      2: blue      3: brown      4: cyan      5: darkblue
6: darkcyan   7: darkgray   8: darkgreen  9: darkorange 10: darkred
11: gold      12: gray      13: green     14: lightcyan 15: lightgray
16: lightgreen 17: magenta   18: maroon    19: navy      20: orange
21: purple    22: red       23: violet    24: yellow    25: transparent
Select Fill Color: 19

Input Border Width [1, 100]: 3
Input the Opacity [0, 1]: 0.75
```

With those inputs the following is a printout of the database entry for that object.

```
Rectangle Anchor Point: (-102, 123) Width: 401 Height: 500 Border: cyan
Fill: navy Border Width: 3 Opacity: 0.75
```

- Option #2: Add a single Square. When the user selects to add a square the following sequence of inputs will come up. Note that these are the same options as the rectangle except that you only need to input a side length in place of a height and width.

```
Choice: 2
Input Upper Left X [-1000, 1000]: 50
Input Upper Left Y [-1000, 1000]: 100
Input Side Length [1, 1000]: 200

1: black      2: blue      3: brown      4: cyan      5: darkblue
6: darkcyan   7: darkgray   8: darkgreen  9: darkorange 10: darkred
11: gold      12: gray      13: green     14: lightcyan 15: lightgray
16: lightgreen 17: magenta   18: maroon    19: navy      20: orange
21: purple    22: red       23: violet    24: yellow    25: transparent
Select Border Color: 16

1: black      2: blue      3: brown      4: cyan      5: darkblue
6: darkcyan   7: darkgray   8: darkgreen  9: darkorange 10: darkred
11: gold      12: gray      13: green     14: lightcyan 15: lightgray
16: lightgreen 17: magenta   18: maroon    19: navy      20: orange
21: purple    22: red       23: violet    24: yellow    25: transparent
Select Fill Color: 5

Input Border Width [1, 100]: 5
Input the Opacity [0, 1]: 1
```

With those inputs the following is a printout of the database entry for that object.


```
Rectangle Anchor Point: (50, 100) Width: 200 Height: 200 Border: lightgreen
Fill: darkblue Border Width: 5 Opacity: 1
```

- Option #3: Add a single Triangle. When the user selects to add a triangle the following sequence of inputs will come up. The first 6 ask for the coordinates of the three vertices. The rest of the options are identical to the square and rectangle.

```
Choice: 3
Input x1 [-1000, 1000]: 23
Input y1 [-1000, 1000]: 45
Input x2 [-1000, 1000]: 62
Input y2 [-1000, 1000]: 55
Input x3 [-1000, 1000]: 107
Input y3 [-1000, 1000]: 45

1: black      2: blue      3: brown      4: cyan      5: darkblue
6: darkcyan   7: darkgray   8: darkgreen  9: darkorange 10: darkred
11: gold      12: gray      13: green     14: lightcyan 15: lightgray
16: lightgreen 17: magenta   18: maroon    19: navy      20: orange
21: purple    22: red       23: violet    24: yellow    25: transparent
Select Border Color: 1

1: black      2: blue      3: brown      4: cyan      5: darkblue
6: darkcyan   7: darkgray   8: darkgreen  9: darkorange 10: darkred
11: gold      12: gray      13: green     14: lightcyan 15: lightgray
16: lightgreen 17: magenta   18: maroon    19: navy      20: orange
21: purple    22: red       23: violet    24: yellow    25: transparent
Select Fill Color: 25

Input Border Width [1, 100]: 2
Input the Opacity [0, 1]: 0.85
```

With those inputs the following is a printout of the database entry for that object.

```
Triangle Points: (23, 45) (62, 55) (107, 45) Border: black
Fill: transparent Border Width: 2 Opacity: 0.85
```

- Option #4: Add a single Circle. When the user selects to add a circle the following sequence of inputs will come up. The first three ask for the coordinates of the center and then the radius of the circle. The rest of the options are identical to the triangle, square and rectangle.

```
Choice: 4
Input Center X [-1000, 1000]: -300
Input Center Y [-1000, 1000]: 200
Input Radius [1, 1000]: 123

1: black      2: blue      3: brown      4: cyan      5: darkblue
6: darkcyan   7: darkgray   8: darkgreen  9: darkorange 10: darkred
11: gold      12: gray      13: green     14: lightcyan 15: lightgray
16: lightgreen 17: magenta   18: maroon    19: navy      20: orange
21: purple    22: red       23: violet    24: yellow    25: transparent
Select Border Color: 6

1: black      2: blue      3: brown      4: cyan      5: darkblue
6: darkcyan   7: darkgray   8: darkgreen  9: darkorange 10: darkred
11: gold      12: gray      13: green     14: lightcyan 15: lightgray
```

```

16: lightgreen  17: magenta    18: maroon     19: navy       20: orange
21: purple     22: red      23: violet     24: yellow     25: transparent
Select Fill Color: 22

```

```

Input Border Width [1, 100]: 7
Input the Opacity [0, 1]: 0.777

```

With those inputs the following is a printout of the database entry for that object.

```

Circle  Center: (-300, 200)  Radius: 123  Border: darkcyan  Fill: red
Border Width: 7  Opacity: 0.777

```

- Options #5–#9: When the user selects to add in any of the randomly generated objects a single input will appear asking for how many items to generate, between 0 and 100. 0 was included in case the user changes their mind and wants to back out.

For example, if the user selects #9, they will see,

```

Number of Objects to Add [0, 100]: 10

```

The database now looks like,

```

Number of Objects in Database = 10

```

```

Rectangle  Anchor Point: (850, 611)  Width: 746  Height: 354  Border: gold
Fill: orange  Border Width: 2  Opacity: 0.187021

```

```

Rectangle  Anchor Point: (28, -525)  Width: 848  Height: 623  Border: purple
Fill: darkcyan  Border Width: 4  Opacity: 0.29898

```

```

Rectangle  Anchor Point: (140, 265)  Width: 296  Height: 731  Border: lightgray
Fill: green  Border Width: 4  Opacity: 0.924671

```

```

Circle  Center: (-28, -494)  Radius: 246  Border: transparent  Fill: brown
Border Width: 5  Opacity: 0.866525

```

```

Rectangle  Anchor Point: (-730, 78)  Width: 560  Height: 560  Border: darkgray
Fill: lightcyan  Border Width: 3  Opacity: 0.347617

```

```

Triangle  Points: (413, 878) (915, 982) (833, 817)  Border: black  Fill: transparent
Border Width: 2  Opacity: 0.592978

```

```

Circle  Center: (341, -857)  Radius: 829  Border: lightgray  Fill: darkblue
Border Width: 1  Opacity: 0.134759

```

```

Circle  Center: (-627, -701)  Radius: 846  Border: red  Fill: blue
Border Width: 3  Opacity: 0.241394

```

```

Rectangle  Anchor Point: (-36, 414)  Width: 237  Height: 237  Border: cyan
Fill: transparent  Border Width: 5  Opacity: 0.400144

```

```

Rectangle  Anchor Point: (-320, -984)  Width: 294  Height: 239  Border: darkred
Fill: yellow  Border Width: 2  Opacity: 0.327422

```

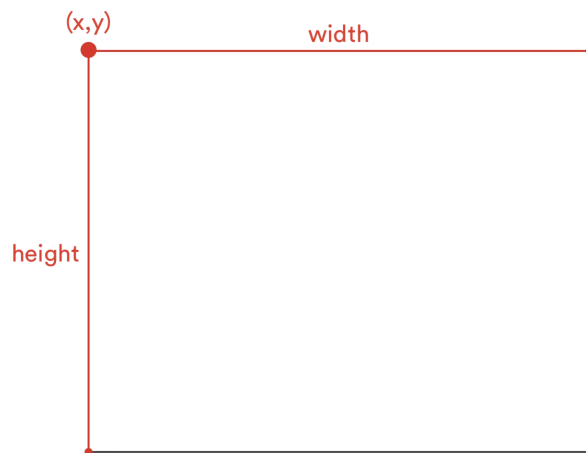
The other random options will create random objects of just the selected type.

4.3 SVG Object and File Syntax

There are far more options and objects that are supported by SVG, we are just going to discuss the ones that will allow you to create an image out of your database. Although there are 4 objects you can create in your database only three SVG objects need to be discussed (since the square and rectangle are both rectangles).

4.3.1 Rectangle & Square

For the rectangle the x and y is the position of the upper left vertex, the width and height are the width and height.



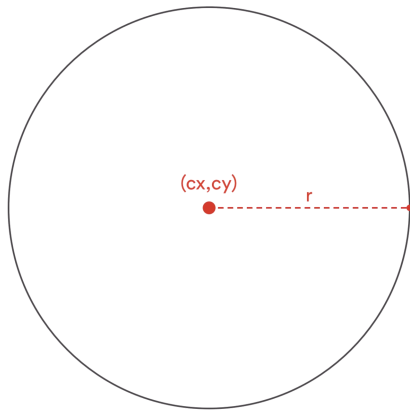
So if you have a rectangle in your database the svgcode function should return a string with the following format.

```
<rect x="850" y="611" width="746" height="354" stroke="gold" fill="orange"
stroke-width="2" opacity="0.187021" />
```

For this one the upper left coordinates are (850, 611), remember this is in pixels, that is dots on the screen. The width and height are 746 and 354 respectively. The border color is gold, the fill color is orange, the border width is 2 and its opacity is 0.187021. With opacity, 0 is completely transparent and 1 is completely opaque. The square will produce the same string except that the height and width will be equal.

4.3.2 Circle

For the circle, the cx and cy is the position of the center, the r is the radius of the circle.



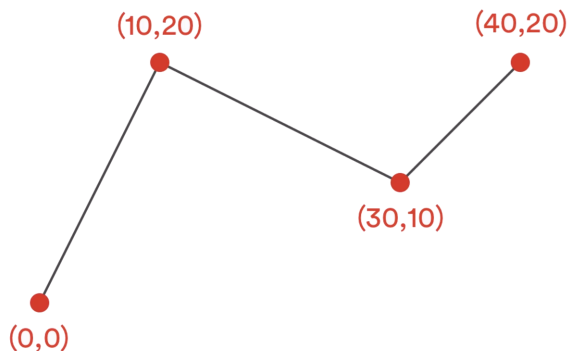
```
<circle cx="-28" cy="-494" r="246" stroke="brown" fill="transparent"
stroke-width="5" opacity="0.866525" />
```

For this one the center coordinates are $(-28, -494)$, and its radius r is 246. The border color is brown, the fill color is transparent, the border width is 5 and its opacity is 0.866525.

4.3.3 Triangle

For the triangle we will use a polyline object, which is a set of connected straight line segments. The points are listed with whitespace between them and the x and y values are separated with a comma.

```
points="0,0 10,20 30,10 40,20"
```



```
<polyline points="413, 878 915, 982 833, 817 413, 878" stroke="black"
fill="transparent" stroke-width="2" opacity="0.592978" />
```

For this one the points are $(413, 878)$, $(915, 982)$, $(833, 817)$, and then the last point is the same as the first to connect the triangle. The border color is black, the fill color is transparent, the border width is 2 and its opacity is 0.592978.

4.3.4 SVG File Structure

The SVG file is simply a text file, no special formats to deal with like jpg or png files. The first lines of the file are,

```
<?xml version="1.0" standalone="no"?>
<svg width="1000" height="1000" viewBox="-998 -996 1993 1993"
version="1.1" xmlns="http://www.w3.org/2000/svg">
```

All of this will not change except for the numbers after the viewBox, we will discuss how to calculate these later. Then you put in each objects svg code that we discussed above. Finally, you simply put </svg> as the last line. An example of a complete svg file is below.

```
<?xml version="1.0" standalone="no"?>
<svg width="1000" height="1000" viewBox="-1052 -661 2600 2600" version="1.1"
xmlns="http://www.w3.org/2000/svg">
<rect x="169" y="379" width="792" height="505" stroke="navy" fill="black"
stroke-width="4" opacity="0.931133" />
<rect x="-965" y="784" width="663" height="663" stroke="blue" fill="transparent"
stroke-width="5" opacity="0.910056" />
<rect x="-809" y="855" width="747" height="235" stroke="lightgreen" fill="darkcyan"
stroke-width="1" opacity="0.855887" />
<rect x="757" y="254" width="791" height="537" stroke="maroon" fill="transparent"
stroke-width="2" opacity="0.193971" />
<rect x="214" y="-169" width="276" height="505" stroke="lightgreen" fill="lightgreen"
stroke-width="5" opacity="0.620652" />
<circle cx="246" cy="-10" r="245" stroke="brown" fill="orange" stroke-width="1"
opacity="0.059643" />
<circle cx="-851" cy="192" r="201" stroke="darkcyan" fill="violet" stroke-width="5"
opacity="0.884747" />
<rect x="-828" y="804" width="123" height="123" stroke="lightgray" fill="darkorange"
stroke-width="1" opacity="0.310103" />
<rect x="169" y="-661" width="835" height="643" stroke="darkgray" fill="violet"
stroke-width="4" opacity="0.234912" />
<circle cx="448" cy="277" r="826" stroke="gold" fill="navy" stroke-width="3"
opacity="0.666855" />
</svg>
```

4.3.5 Calculating the View Box Values

The only function in the shape hierarchy we did not discuss yet is the purely virtual getBounds function. Recall that this returns a pointer to a dynamic array of four integer entries. So getBounds will allocate the memory for the array and populate the array with the bounding box for the object. It will pass this pointer to the main that will calculate the view box values and then the main will delete the array. The bounding box for an object is the smallest rectangle that completely contains the object. The getBounds array is to store the minimum x , maximum x , minimum y , and maximum y values for the bounding box in that order. Hence this is the smallest rectangle bounds that contains each object.

Now to find the values for the view box. First find the minimum x , maximum x , minimum y , and maximum y values for all objects in the scene. For example, say your bounding boxes are

```
[-12, 24, -4, 17]
[  6, 72, 15, 50]
[-23, 50,  0, 71]
[ 10, 20, 10, 20]
```

Then the scene minimums and maximums are `[-23, 72, -4, 71]`. Then find the height and width of the total bounding box from the scene minimums and maximums. So the width would be $72 - (-23) = 95$ and the height would be $71 - (-4) = 75$. Take the maximum of the width and the height, in this case 95. Now the view box values are the scene minimum x , the scene minimum y , and the max of the height and width for the view boxes height and width. In this example it would be `viewBox="-23 -4 95 95"`.

4.4 More Details for the Application

The following are some additional requirements of the program.

4.4.1 User Friendliness

We would like to make this program as user friendly as possible. One thing we can do at this stage is use exception handling to take care of determining data types and data ranges that the user types in and in the case where it is incorrect we can ask for the input again. One thing we do not want is to ask for a number and if the user types in a string have the program crash.

In the example code, in the `CPP_Modules` directory, in the `Exceptions` directory, in the `SystemExceptions` directory, in the `main.cpp` file there is a function `InputInteger` that catches system level exceptions from the `stoi` and `stod` functions. This allows us to catch input errors from the user. For example, in a run of the program in the input of a rectangle I tried the following.

[illegible]

Some of these inputs would have crashed the program if we were just using a `cin`. But with the use of this `InputInteger` function we can keep the program from crashing. Take the `InputInteger` function from the example and put it into your program. Update it as follows. Include three parameters, a string prompt, a long integer lower bound and long integer upper bound. The string parameter prompt should replace the "Input an integer: "

prompt in the function, the lower and upper bounds are to be the smallest and largest acceptable values for the input. If the input is out of range there should be a message about that like the one in the example above, and of course ask for the input again.

Now create a similar function `InputDouble` that does the same thing as `InputInteger` but the bounds are doubles and the return type is a double. All user input is to be done through these two functions.

4.4.2 Database Data Structure

The database holding all of the shapes is to be your final version of the doubly linked list. Although this is obvious since we are using polymorphism, the linked list will store `Shape` pointers which point to each specific shape object in the database. If you were not able to get a working version of the doubly linked list you may use the STL list to do the storage but you will not earn full credit for this portion of the project.

4.4.3 Modularity

Break the program up into nice modular sections. Functions should be focused on small particular tasks. Minimize the amount of code duplication and utilize reuse where possible. Keep the code easily readable and revise any sections that are too cryptic.

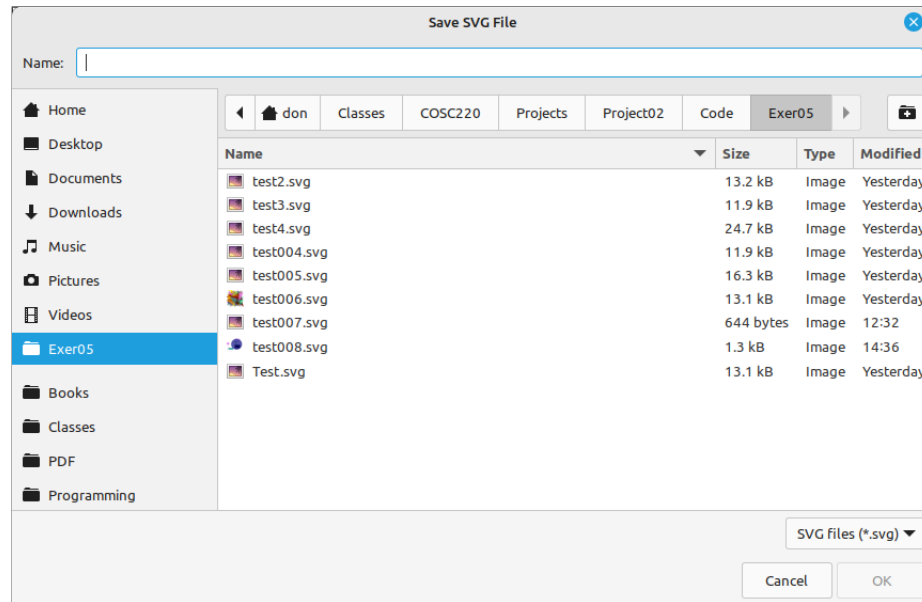
4.4.4 Memory Management

There is a lot of dynamic memory allocation and memory management going on in this application. Make sure that there are no memory errors, in reading, writing, jumping, etc. as well as no leaks.

4.5 Extra Credit: Dialog Boxes

For some extra credit, this is purely optional, incorporate a save dialog box for saving the svg file.

In the example code, in the `CPP_Modules` directory, in the `TinyFileDialogs` directory, there are two files `tinyfiledialogs.h` and `tinyfiledialogs.cpp` that contain code for you to easily bring up the standard system dialog boxes for opening files, saving files, selecting colors, message boxes, etc. There is also a readme file showing the general syntax for the functions and a short `main.cpp` example program using these dialogs. If you incorporate these then when the user selects option 12, instead of typing in a filename they would see,



They would then select a directory for storage and input a filename and click OK. At this point the program will save the svg file. If the user selects Cancel then no file will be saved. As with the text input route, if the extension is not “.svg” add it to the filename. For this dialog box you should be selecting only single files and the filter should be set to one file type, SVG Files with an extension of .svg.

