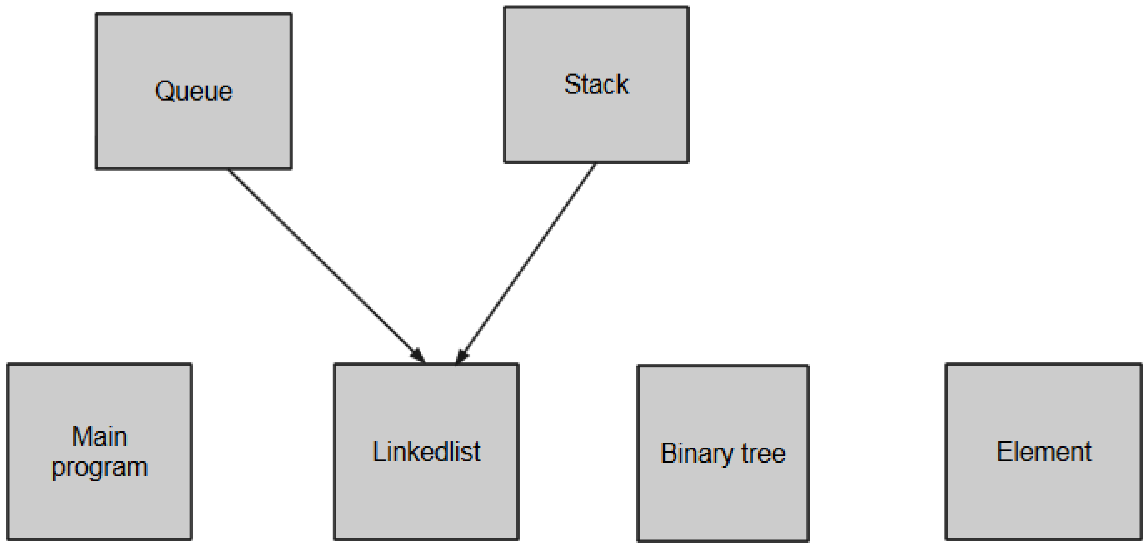
Rube Goldberg Machine

1. ADTs design



The diagram above describes the interdependence between my ADTs. Queue and stack make use of linkedlist. The element class is used to instantiate objects whose attributes are firstname, last name, age and birthdate. The objects instantiated from this class are used to construct nodes in a linkedlist and binary tree.

1.2 Operations hidden within my ADTs and performance analysis of these operations in terms of time complexity

1.2.1 Linkedlist

Void append(Element*)

This function creates a linkedlist where a node is appended at the end of the list. It was used to perform the queuing operation and the time complexity of adding a node to the list is o(n) where n is the number of nodes in the list.

Void append (Element*)

This function creates a linkedlist where a node is added at the head of the list. It was used to create a stack. The time complexity of adding a node to the list is o(1), that is to say it takes constant time to add a node to the list irrespective to its size.

Element* removeFirst()

This function returns a pointer to an object of type element. It was used to perform the dequeuing operation.

Element* removeLast()

This function returns a pointer to an object of type element. It was used to empty the stack.

Void sort (Element object [], int begin, int end)

This function is used to sort elements in a linkedlist, it takes three parameters: an array of objects of type element, the beginning and end of the array. This function is used recursively and its time complexity is o(n) where n is the number of elements in the array.

Int getPartitionIndex (Element object [], int begin, int end)

This function returns the index position of the pivot and it is invoked in a recursive manner inside the sort function. The pivot is an element in the array that is randomly chosen, in my case the pivot is the last element in the array. The idea behind the pivot is that when this function returns we are sure that all the elements smaller than the pivot are on the left-hand side of the pivot and all elements greater than the pivot are on the right-hand side of the pivot. All the operations are performed inside the initial array, no need to create another array whenever the function is invoked, which is quite efficient in terms of memory usage. The time complexity of this function is o(n) where n is the number of elements in the array.

Void swap (Element object [], int index1, int index2)

This function swaps two elements at two different index positions in the array. It is invoked inside the getPartitionIndex function. The swapping operation is required to push all the elements lower than the pivot to the left-hand side of the pivot and all the elements greater than the pivot to the right-hand side of the pivot.

1.2.2 Queue

Void enqueue (Element*)

This function does nothing other than invoking the append function in the linkedlist class.

Element* dequeue ()

This function does nothing other than invoking the removeFist function in the linkedlist class.

1.2.3 Stack

Void push (Element*)

This function does nothing other than invoking the preappend function in the linkedlist.

Element* pop ()

This function does nothing other than invoking the removeLast function in the linkedlist class.

1.2.4 Binary tree

Node* getNode(Element*)

This function returns a pointer to the newly created node in the binary tree. A node in the binary tree comprises three fields: an object of type element, two pointers used to construct the left and right sub-tree.

Node* add (Node* root, Element* object)

This function returns a pointer to the root node in the binary tree. It takes two parameters: a pointer to the root node and an object of type element. In this function, nodes are recursively added to the tree.

Void startPreorderTraversal (Node* root)

This function receives a pointer to the root node in the binary tree as an argument and then performs the preorder traversal in a recursive fashion. The time complexity of this function is o(n) where n is the number of nodes in a binary tree.

Void startInorderTraversal(Node* root)

This function receives a pointer to the root node in the binary tree as an argument and then performs the inorder traversal in a recursive fashion. The time complexity of this function is o(n) where n is the number of nodes in a binary tree.

Void startPostorderTraversal(Node* root)

This function receives a pointer to the root node in the binary tree as an argument and then performs the postorder traversal in a recursive fashion. The time complexity of this function is o(n) where n is the number of nodes in a binary tree.

1.3 Instructions on how to use my program

All my codes were written using the codeBlocks IDE with GNU GCC compiler. It is quite straightforward to use my program, all you have to do is to follow the following steps:

- Put the folder named “assignment4” in a particular location on your computer. This folder contains all the source, header files and the executable file as well as the input files.
- Launch your favorite C++ compiler, go to file menu, choose open and browse to the assignment4 folder.
- Select the C++ assignment4 project
- Compiler and run the program.

Expected outputs:

6 text files will be created and placed in the assignment4 folder; these files are named output1.dat through output6.dat. The content of these files is what the assignment states they should contain.

Note: the program creates 6 output text files instead of 7 as it is stated in the assignment. This is because the output6.dat file contains the sorted list of element objects from both the InputFile and more-input text files. After running the program, the first lines of the output6.dat file will contain single quotation marks, but despite this, just scroll down and you will see a sorted list of object elements from both the InputFile and more-input text files. I tried to solve this issue, but I couldn't. I think the reason why this happened, it's because I used a loop to loop through the array of sorted elements and this loop does more iterations than it is supposed to do. It's hard to get the size of an array in C++, you can't use size() or length() as it is in Java unless you use the vector container.

1.4 Analysis of the application in terms of complexity and correctness

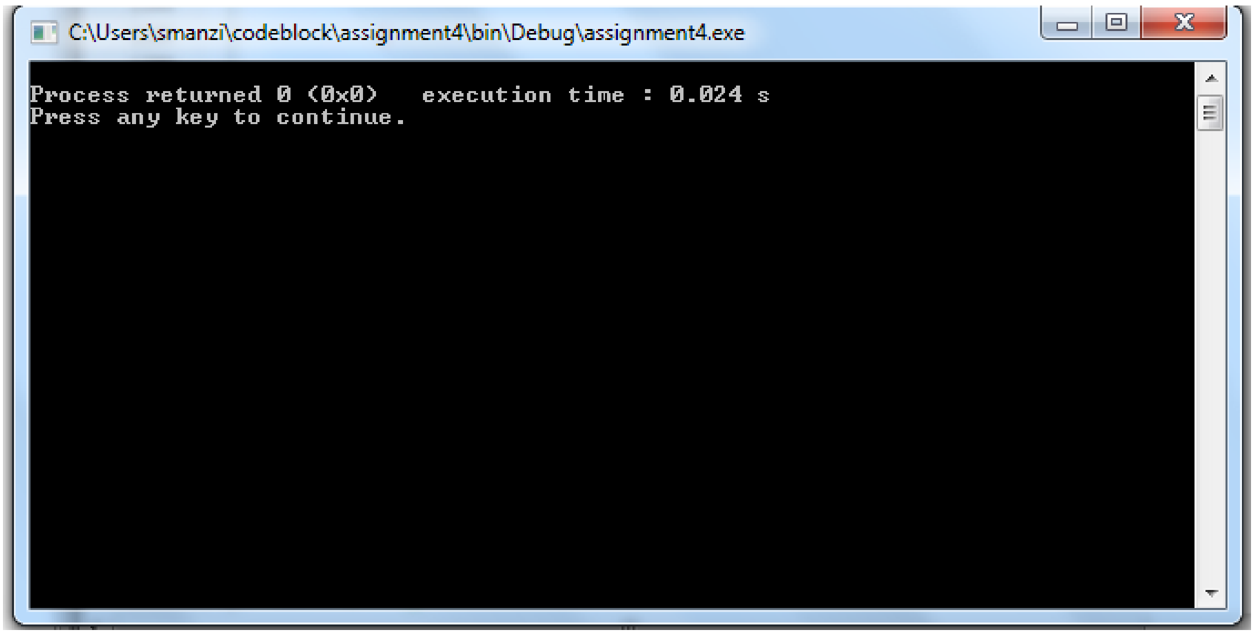
1.4.1 Memory usage

This application is memory efficient due to the following reasons:

- Code are reused to implement some of the ADTs such as queue and stack.
- Recursive approach was taken to implement some of the subroutines, which saves a bunch of code lines.

1.4.2 Time complexity

Apart from the time complexity of individual subroutines discussed above, the execution time of the whole program is within a few milliseconds.

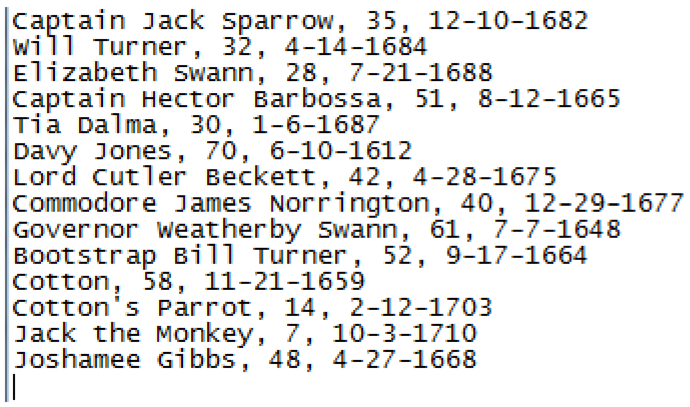

C:\Users\smanzi\codeblock\assignment4\bin\Debug\assignment4.exe

```
Process returned 0 (0x0)   execution time : 0.024 s
Press any key to continue.
```

1.4.2 Correctness

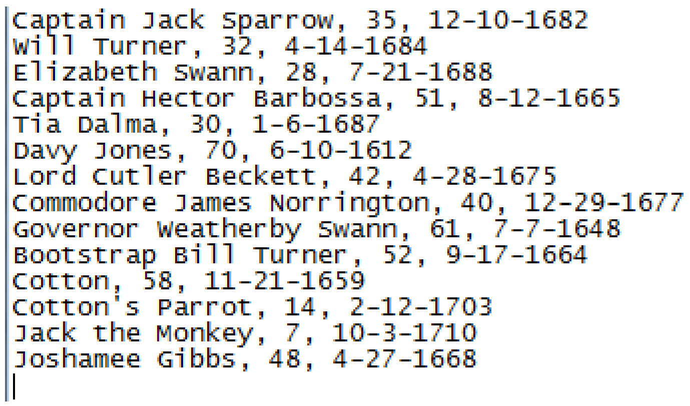
I decided to discuss the correctness of my application in terms of how the program works with respect to the expected outputs. Below is the content of the input file and output files generated at each stage of the application.

Contents of the input file



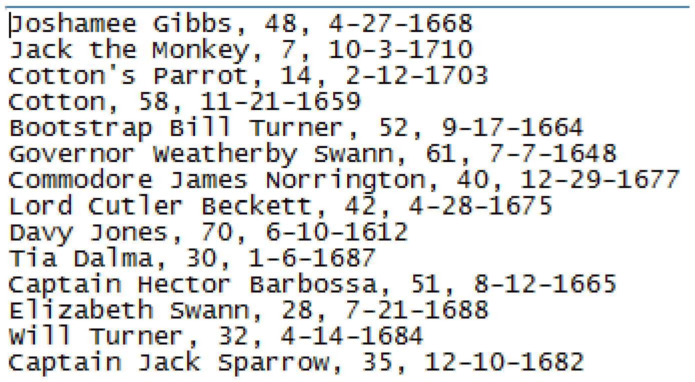
```
Captain Jack Sparrow, 35, 12-10-1682
Will Turner, 32, 4-14-1684
Elizabeth Swann, 28, 7-21-1688
Captain Hector Barbossa, 51, 8-12-1665
Tia Dalma, 30, 1-6-1687
Davy Jones, 70, 6-10-1612
Lord Cutler Beckett, 42, 4-28-1675
Commodore James Norrington, 40, 12-29-1677
Governor Weatherby Swann, 61, 7-7-1648
Bootstrap Bill Turner, 52, 9-17-1664
Cotton, 58, 11-21-1659
Cotton's Parrot, 14, 2-12-1703
Jack the Monkey, 7, 10-3-1710
Joshamee Gibbs, 48, 4-27-1668
```

Contents of the output1.dat file (created after dequeuing the queue)



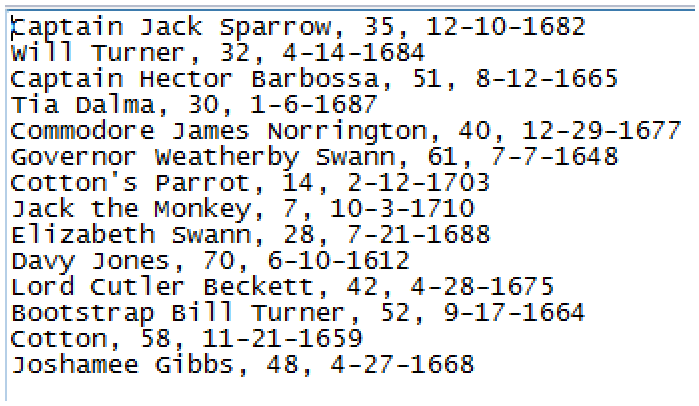
```
Captain Jack Sparrow, 35, 12-10-1682
Will Turner, 32, 4-14-1684
Elizabeth Swann, 28, 7-21-1688
Captain Hector Barbossa, 51, 8-12-1665
Tia Dalma, 30, 1-6-1687
Davy Jones, 70, 6-10-1612
Lord Cutler Beckett, 42, 4-28-1675
Commodore James Norrington, 40, 12-29-1677
Governor Weatherby Swann, 61, 7-7-1648
Bootstrap Bill Turner, 52, 9-17-1664
Cotton, 58, 11-21-1659
Cotton's Parrot, 14, 2-12-1703
Jack the Monkey, 7, 10-3-1710
Joshamee Gibbs, 48, 4-27-1668
```

Contents of the output2.dat file (created after dequeuing the queue which contained element objects from the stack)



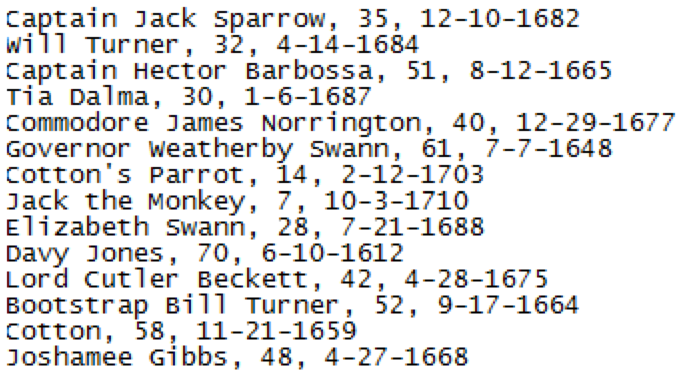
```
Joshamee Gibbs, 48, 4-27-1668
Jack the Monkey, 7, 10-3-1710
Cotton's Parrot, 14, 2-12-1703
Cotton, 58, 11-21-1659
Bootstrap Bill Turner, 52, 9-17-1664
Governor Weatherby Swann, 61, 7-7-1648
Commodore James Norrington, 40, 12-29-1677
Lord Cutler Beckett, 42, 4-28-1675
Davy Jones, 70, 6-10-1612
Tia Dalma, 30, 1-6-1687
Captain Hector Barbossa, 51, 8-12-1665
Elizabeth Swann, 28, 7-21-1688
Will Turner, 32, 4-14-1684
Captain Jack Sparrow, 35, 12-10-1682
```

Contents of the output3.dat file (created after doing preorder traversal of the binary tree)



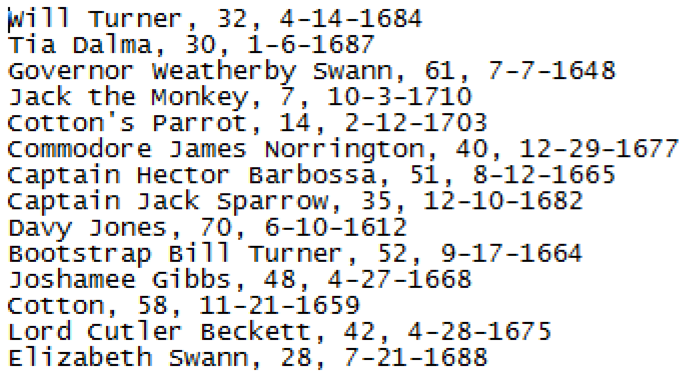
```
Captain Jack Sparrow, 35, 12-10-1682
Will Turner, 32, 4-14-1684
Captain Hector Barbossa, 51, 8-12-1665
Tia Dalma, 30, 1-6-1687
Commodore James Norrington, 40, 12-29-1677
Governor Weatherby Swann, 61, 7-7-1648
Cotton's Parrot, 14, 2-12-1703
Jack the Monkey, 7, 10-3-1710
Elizabeth Swann, 28, 7-21-1688
Davy Jones, 70, 6-10-1612
Lord Cutler Beckett, 42, 4-28-1675
Bootstrap Bill Turner, 52, 9-17-1664
Cotton, 58, 11-21-1659
Joshamee Gibbs, 48, 4-27-1668
```

Contents of the output4.dat file (created after doing post-order traversal of the binary tree)



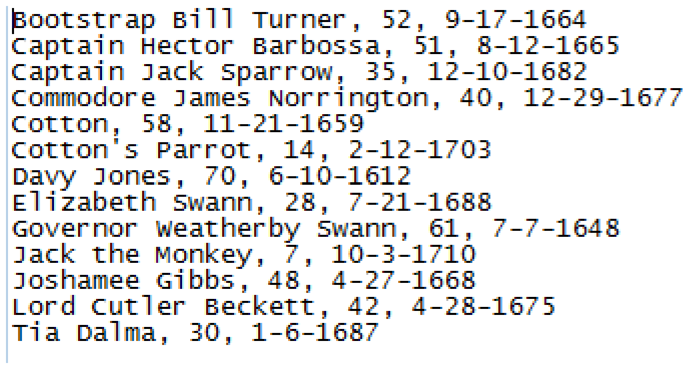
```
Captain Jack Sparrow, 35, 12-10-1682
Will Turner, 32, 4-14-1684
Captain Hector Barbossa, 51, 8-12-1665
Tia Dalma, 30, 1-6-1687
Commodore James Norrington, 40, 12-29-1677
Governor Weatherby Swann, 61, 7-7-1648
Cotton's Parrot, 14, 2-12-1703
Jack the Monkey, 7, 10-3-1710
Elizabeth Swann, 28, 7-21-1688
Davy Jones, 70, 6-10-1612
Lord Cutler Beckett, 42, 4-28-1675
Bootstrap Bill Turner, 52, 9-17-1664
Cotton, 58, 11-21-1659
Joshamee Gibbs, 48, 4-27-1668
```

Contents of the output5.dat file (created after doing inorder traversal of the binary tree)



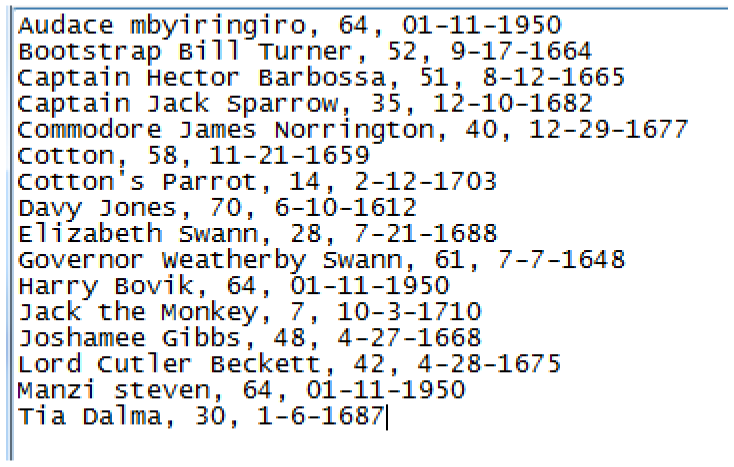
```
Will Turner, 32, 4-14-1684
Tia Dalma, 30, 1-6-1687
Governor Weatherby Swann, 61, 7-7-1648
Jack the Monkey, 7, 10-3-1710
Cotton's Parrot, 14, 2-12-1703
Commodore James Norrington, 40, 12-29-1677
Captain Hector Barbossa, 51, 8-12-1665
Captain Jack Sparrow, 35, 12-10-1682
Davy Jones, 70, 6-10-1612
Bootstrap Bill Turner, 52, 9-17-1664
Joshamee Gibbs, 48, 4-27-1668
Cotton, 58, 11-21-1659
Lord Cutler Beckett, 42, 4-28-1675
Elizabeth Swann, 28, 7-21-1688
```

Contents of the output6.dat file (created after sorting the elements in a linkedlist)



```
Bootstrap Bill Turner, 52, 9-17-1664
Captain Hector Barbossa, 51, 8-12-1665
Captain Jack Sparrow, 35, 12-10-1682
Commodore James Norrington, 40, 12-29-1677
Cotton, 58, 11-21-1659
Cotton's Parrot, 14, 2-12-1703
Davy Jones, 70, 6-10-1612
Elizabeth Swann, 28, 7-21-1688
Governor Weatherby Swann, 61, 7-7-1648
Jack the Monkey, 7, 10-3-1710
Joshamee Gibbs, 48, 4-27-1668
Lord Cutler Beckett, 42, 4-28-1675
Tia Dalma, 30, 1-6-1687
```

Contents of the output7.dat file (created after inserting more entries in the sorted list).



```
Audace mbyiringiro, 64, 01-11-1950
Bootstrap Bill Turner, 52, 9-17-1664
Captain Hector Barbossa, 51, 8-12-1665
Captain Jack Sparrow, 35, 12-10-1682
Commodore James Norrington, 40, 12-29-1677
Cotton, 58, 11-21-1659
Cotton's Parrot, 14, 2-12-1703
Davy Jones, 70, 6-10-1612
Elizabeth Swann, 28, 7-21-1688
Governor Weatherby Swann, 61, 7-7-1648
Harry Bovik, 64, 01-11-1950
Jack the Monkey, 7, 10-3-1710
Joshamee Gibbs, 48, 4-27-1668
Lord Cutler Beckett, 42, 4-28-1675
Manzi steven, 64, 01-11-1950
Tia Dalma, 30, 1-6-1687
```