Read Me:

1. PA1
2. Connor Pham (template from Professor Manju)
3. PA1.cpp, ReadMePhamPA#1.pdf
4. One may run PA1.cpp with a C++ compiler/IDE such as Clion. To install go to jetbrains.com to learn more about installation. For the ReMePhamPA#1.pdf, this can be read via the web. If one wants to be able to make edits, a google account is required to access the google doc. Create a google account and gain access to the document if one wants to "run" and edit the file.
5. Only the main method requires user input, which are integers to navigate through the menu in the console. Upon entering an integer from the given choices switch cases will proceed to do the various operations on the list. In cases 2,3,4,7, and 10, there are 1-2 extra (int) inputs needed to either select what data they would like to insert and/or to select the index at which they want to delete a node and its data.

   If the ***user inputs "1"*** then the delete list destructor is called to delete the list. Program is exited.

   If the ***user inputs "2"*** they will be prompted to enter another integer 1-5 that corresponds to extra Data objects (d11-d15) that are part of the data class. The result calls the bool insert() method to insert data at the head position. This method allows edits to the list to have T* value inserted at an index within bounds of the list. Finally printList() is called, which prints the most current list after modifications.

   If the ***user inputs "3"*** they will be prompted to enter another integer 1-5 that corresponds to extra Data objects (d11-d15) that are part of the data class. The result calls the bool insert() method to insert data at the tail position. This method allows edits to the list to have T* value inserted at an index within bounds of the list. Finally printList() is called, which prints the most current list after modifications.

If the ***user inputs "4"*** they will be prompted to enter 2 integers, one integer between 0-9 (indexes of original list) and an integer between 1-5 that corresponds to extra Data objects (d11-d15) that are part of the data class. The result calls the bool insert() method to insert data at the position of what the first input was. This method allows edits to the list to have T* value inserted at an index within bounds of the list. Finally printList() is called, which prints the most current list after modifications.

If the ***user inputs "5"*** the deleteAtHead() method is called which deletes the head node and repoints the nodes accordingly. This method calls printList() which prints the most current list after modifications.

If the ***user inputs "6"*** the deleteAtTail() method is called which deletes the tail node and repoints the nodes accordingly. This method calls printList() which prints the most current list after modifications.

If the ***user inputs "7"*** they are prompted to input another integer from 0-9 (indexes of original list), to determine where a node will be deleted. This method calls printList() which prints the most current list after modifications.

If the ***user inputs "8"*** the reverseList() method is called which reassigns the pointers to point to the nodes in a reverse order. This method calls printList() which prints the most current list after modifications.

If the ***user inputs "9"*** the sortList() method is called, while loops to bubble sort each node's data. It uses a helper function called compareData(const Data & other) which is part of the data class. This is used to compare the integer value of two nodes that acts as a x < y inequality. sortList() calls printList() which prints the most current list after modifications.

If the ***user inputs "10"*** they will be prompted to input another integer within the original length of the list (1-10), to determine what corresponding Data object they would like to search for multiples of. This calls the countMultiples(T value) method which calls the sortList() method to then iteratively check each node's data for a match as T value. During the iterations, a helper function called compareEqualData(const Data & other) which is part of the data class that makes a comparison between ints like x==y. Everytime this condition is met, count increments. Then printList() is called which prints the most current list after modifications. Then the method returns count (amount of duplicates).

If the ***user inputs "11"*** the removeMultiples() method is called, within that the sortList() method is used. During the iterations, a helper function called compareEqualData(const Data & other) which is part of the data class that makes a comparison between ints like x==y. Everytime this condition occurs the temp variable is deleted containing the repeated node with the same data. Then printList() is called which prints the most current list after modifications.

If the ***user inputs "12"*** the headTailSplit() method is called where two separate lists are created and half go into list A and the other half list B. In the case of uneven list length, list B has the greater half. The original list is deleted. This method calls printList() which prints the two new lists after modifications. Program is exited.

If the ***user inputs "13"*** the program is exited.

6. Big O functionality: Deriving the Big O of each function as they appear chronologically in the code portion.

```
Data(int value, string name){} - O(1); All operations are constant time.
void print(){}- O(1); All operations happen in constant time.
bool compareData(const Data & other) {} - O(1); All operations happen in
constant time.
bool compareEqualData(const Data & other) {} - O(1); all operations happen in
constant time.
void print(){}- O(1); All operations happen in constant time.
DoubleLinkedList(T *value){} - O(1);All operations happen in constant time.
~DoubleLinkedList() {} - O(n); This is a while loop that runs n times depending
on the amount of nodes in the list.
void printList() {} - O(n); There is a while loop that runs n times depending
on the amount of nodes in the list.
Node<T> *getHead(){} - O(1); All operations are assignments and conditionals
are fixed that run in constant time.
Node<T> *getTail(){} - O(1); All operations are assignments and conditionals
are fixed that run in constant time.
int getLength() {} - O(1); Simple print and return statement that is a constant
time operation and fixed.
Node<T> *get(int index) {} - O(n); Longest operation is the for loop that runs
n times based on the index parameter.
bool set(int index, T *value) {} - O(1); Fixed amount of operations that are
just assignments/conditionals.
void append(T *value) {}- O(1); Operations are fixed, the longest being an
assignment statement.
void prepend(T *value) {}- O(1); Operations are fixed, the longest being an
assignment statement.
```

bool insert(int index, T *value) {} O(1); All statements are fixed and run regardless of list size. All operations are of constant time.

void deleteAtHead() {} - O(n); All operations are fixed and run at constant time, but the printList() method is called and that has a for loop that runs n times based on the number of nodes in the list.

void deleteAtTail() {} - O(n); All operations are fixed and run at constant time, but the printList() method is called and that has a for loop that runs n times based on the number of nodes in the list.

void deleteAtIndex(int index) {} - O(n); All operations are fixed and run at constant time, but the printList() method is called and that has a for loop that runs n times based on the number of nodes in the list.

void sortList() {} - O(n^2); The longest operation is the nested while loop inside the do while loop. The loop goes n times based on the condition (boolean) and the other loop goes n times based on the amount of nodes in the list. Since it's nested, they multiply and the worst time complexity is n^2.

void removeMultiples(){} - O(n^2); This method has a while loop that goes n times but since it calls the sortList() method so the worst time complexity is n^2, since sortList() runs to a O(n^2) and supersedes the other loop.

int countMultiples(T value) {} - O(n^2)-This method has a while loop that goes n times but since it calls the sortList() method so the worst time complexity is n^2, since sortList() runs to a O(n^2) supersedes the other loop.

void headTailSplit() {} - O(n);All operations in this method are constant time, except the while loop that iterates n times based on the count variable, therefore, the worst time complexity is O(n). Also the other loop runs n times.

void reverseList() {} - O(n); The while loop runs n times based on the number of nodes are in the list.