CSC 228 Data Structures and Algorithms
Fall 2017
Instructor: Dr. Natarajan Meghanathan

Project 1 (List ADT - Dynamic Arrays)

Due: September 13, 2017 @ 1 PM

Consider the implementation of the List class using dynamic arrays as discussed in the class/ slides/ lecture code. Extend the code for the List class with each of the following functions. Each function should be implemented independent of each other. Analyze the asymptotic time complexity of each of the implementations with 'n' considered as the size of the List.

(i) Add a member function called **print( )** to the List class that prints the contents of the List from index 0 to endOfArray with a space between each element.

```
void print( ){
.....
}
```

(ii) Add a member function called **subList(int fromIndex, int toIndex)** to the List class that returns an object of class List itself. The returned List object is a sub list of the List object on which the subList function is called. The subList should basically create a new List object, copy the contents of the List (fromIndex to toIndex) on which the function is called to the new List and return the sub list to the main function. You should not create any other function and just make use of one or more of the existing functions in the current version of the List class.

```
List subList(int fromIndex, int toIndex ){
.........
}
```

(iii) Add a member function called **rotateList(int rotationSpace)** to the List class that rotates the elements in the List by 'rotationSpace' number of elements to the left. For example, if the contents of the List are (before rotation): 10 12 5 7 19 21, then if the List is rotated to the left with a rotationSpace of 2, then the rotated List should be: 5 7 19 21 10 12. Note that the original List should be updated with the rotated contents so that after calling the rotateList function on the original List object, if one were to print the contents of the original List object, the rotated contents should be printed.

```
void rotateList(int rotationSpace){
...........
}
```

(iv) Add a member function called **reverseList( )** to the List class that reverses the contents of the list. For example if the contents of the List are (before reverse): 10 12 5 7 19 21, then after calling the reverseList function, the contents of the List should be updated as: 21 19 7 5 12 10. You should implement the reverseList function without using any additional memory or temporary array that is proportional to the size of the list. Just one temporary variable (or in other words, a constant amount of additional memory that is independent of the size of the list, denoted O(1) memory) should be sufficient.

```
void reverseList( ){
.........
}
```

(v) Add a member function called **segregateEvenOdd( )** to the List class that segregates the contents of the list into a sequence of even integers followed by odd integers. The order of appearance of the even integers (and odd integers) could be different from that in the original list. For example, if the contents of the List are (before segregation into even and odd): 12 34 45 9 8 90 3, then the contents of the List after calling the segregateEvenOdd function could be: 12 34 90 8 9 45 3. Again, you should implement the **segregateEvenOdd( )** function by using only O(1) additional memory.

```
void segregateEvenOdd( ){
.....
}
```

Submission

(i) Report: The complete code for the List class containing all the functions (included in the original code as well as the code for the functions implemented in this project) and your time complexity and space complexity analysis for each of the functions implemented. You should also include a snapshot of the execution of your program for list size chosen from 10 to 15. A sample screenshot of the expected snapshot is shown below.

(ii) Video: A video explaining the implementations for each of the above functions as well as illustrating the execution of the code for a sample list size. You could record the video using a desktop recording software offline and upload to Canvas or use the recording software available in Canvas.

Both the report and video should be submitted through the project link in Canvas by 1 PM on Sept. 13.

Sample Screenshot of Execution

```
Enter list size: 9
Enter element # 0 : 12
Enter element # 1 : 14
Enter element # 2 : 15
Enter element # 3 : 13
Enter element # 4 : 16
Enter element # 5 : 17
Enter element # 6 : 18
Enter element # 7 : 19
Enter element # 8 : 24
Original List: 12 14 15 13 16 17 18 19 24
Sub list from 2 to 4: 15 13 16
After rotation (2 elements to the left): 15 13 16 17 18 19 24 12 14
After reverse: 14 12 24 19 18 17 16 13 15
After even/odd segregation: 14 12 24 16 18 17 19 13 15
```

C++ main function

The main function for testing your implementations is as follows. Your implementations of the functions for the List class should run when called from the main function.

```
int main(){

	int listSize;

	cout << "Enter list size: ";
	cin >> listSize;

	List integerList(1); // we will set the maxSize to 1 and double it as and when needed

	for (int i = 0; i < listSize; i++){

		int value;
		cout << "Enter element # " << i << " : ";
		cin >> value;

		integerList.insertAtIndex(i, value);
	}


	cout << endl;

	cout << "Original List: ";
	integerList.print();

	List sublist = integerList.subList(2, 4);
	cout << "Sub list from 2 to 4: ";
	sublist.print();


	integerList.rotateList(2);
	cout << "After rotation (2 elements to the left): ";
	integerList.print();


	integerList.reverse();
	cout << "After reverse: ";
	integerList.print();

	integerList.segregateEvenOdd();
	cout << "After even/odd segregation: ";
	integerList.print();

return 0;
}
```

Java main function

The main function (and the class) for testing your implementation is as follows. Your implementations of the functions for the List class should run when called from the main function.

```
class DynamicListArray{

	public static void main(String[] args){

	int listSize;

	Scanner input = new Scanner(System.in);

	System.out.print("Enter list size: ");
	listSize = input.nextInt();

	List integerList = new List(1); // we will set the maxSize to 1 and double it as and when needed

	for (int i = 0; i < listSize; i++){

		int value;

		System.out.print("Enter element # " + i + " : ");
		value = input.nextInt();

		integerList.insertAtIndex(i, value);
	}


	System.out.print("Original List: ");
	integerList.print();

	List sublist = integerList.subList(2, 4);
	System.out.print("Sub list from 2 to 4: ");
	sublist.print();

	integerList.rotateList(2);
	System.out.print("After rotation (2 elements to the left): ");
	integerList.print();


	integerList.reverse();
	System.out.print("After reverse: ");
	integerList.print();

	integerList.segregateEvenOdd();
	System.out.print("After even/odd segregation: ");
	integerList.print();

	}

}
```