Assignment 1: List

You need to implement a List ADT as defined in Table 1. There is no restriction on the programming language.

Table 1: List ADT definition.

Fn #	Function	Param.	Ret.	After functions execution	Comment
	[before function execution]			<20, 23 | 12, 15>	The vertical bar indicates the current position. Here the current position is 2 (first position is 0) and corresponds to the element 12.
1	clear()	-		<>	Clear contents from the list, to make it empty. <> means an empty list.
2	insert(item)	19		<20, 23 | 19, 12, 15>	Inserts an element at the current location.
3	append(item)	19		<20, 23 | 12, 15,19>	Appends an element at the end of the list.
4	remove()	-	12	<20, 23 | 15>	Remove and return the current element.
5	moveToStart()	-		< | 20, 23, 12, 15>	Set the current position to the start of the list
6	moveToEnd()	-		<20, 23, 12 |15>	Set the current position to the end of the list
7	prev()	-		<20 | 23, 12, 15>	Move the current position one step left. No change if already at the beginning.
8	next()	-		<20, 23 , 12 | 15>	Move the current position one step right. No change if already at the end.

Fn #	Function	Param.	Ret.	After functions execution	Comment
9	length()	-	4	<20, 23 | 12, 15>	Return the number of elements in the list.
10	currPos()	-	2	<20, 23 | 12, 15>	Return the position (in the list) of the current element.
11	moveToPos(int pos)	1		<20 | 23, 12, 15>	Set current position.
12	getValue()	-	12	<20, 23 | 12, 15>	Return the current element.
13	Search(item)	23	1	<20, 23 | 12, 15>	Search returns the position of the element item or -1 if not found.
13	Search(item)	29	-1	<20, 23 | 12, 15>	

The implementation must support the various types of 'elements'. Use templates in C++ or something equivalent in the programming language you intend to use. You need to provide two different implementations, namely, Array Based (Arr) and Linked List (LL) Based Implementations. The size of the list is only limited by the memory of the computing system, i.e., in case of Arr implementation, there must be a way to dynamically grow the list size as follows: the list should **allocate memory dynamically** in chunk of X elements, i.e., initially the list should be able to hold X elements; as soon as the attempt is made to insert the X+1th element, memory should be allocated such that it can hold 2X elements and so on. Static implementation would result in deduction of marks.

Appropriate constructors and destructors must be implemented. Constructors should at least support initializing (a) an empty list (b) a list with Y elements where Y can be provided in an array in order (for Arr implementation Y < X). You may implement extra helper functions but those will not be available for programmers to use. So, while using the list implementations, one can only use the methods listed in Table 1. Please note that during evaluation, identical main function will be used to check the functionality of both Arr and LL implementations. Also write a simple main function to demonstrate that all the functions are working properly, for both implementations. Recall that, the same main function should work for both the implementation except for the object instantiation part. Please follow the following input format.

Input format (for checking the Arr and LL implementations):

One line containing the initial length of the list and the memory chunk size, space separated: K, X (K < X). For LL implementation X will be ignored.

One line containing K numbers, space separated, to be initialized the list with K elements.

Several lines (indicating the tasks to perform on the list), each containing two integers, Q (Fn #), 0<=Q<=13 and P (parameter). For each line, the program will execute Fn # Q with parameter P. If the respective function does not take a parameter, P will be ignored. If Q is 0, the program will exit.

Example input:

```
5 10
10 12 34 9 1
11 3
4 0
7 0
2 20
9 0
6 0
3 99
8 0
5 0
13 12
12 0
10 0
1 0
0 0
```

Output Format:

At first the system will output in one line the items of the list within angle bracket, space separated identifying the current position using "I" as defined/shown in Table 1. Then for each task, it will output two lines: the first line will output the return value (if available) or -1 (if no return value is expected); the second line will output the the the items of the list, space separated identifying the current position using "I" as defined/shown in Table 1.

Expected Output of the example input:

```
<| 10 12 34 9 1>
<10 12 34 | 9 1>
-1
<10 12 34 | 1>
9
```

```
<10 12 | 34 1>
-1
<10 12 | 20 34 1>
-1
<10 12 | 20 34 1>
5
<10 12 20 34 | 1>
-1
<10 12 20 34 | 1 99>
-1
<10 12 20 34 1 | 99>
-1
<| 10 12 20 34 1 99>
-1
<| 10 12 20 34 1 99>
1
<| 10 12 20 34 1 99>
10
<| 10 12 20 34 1 99>
0
<>
-1
```

Using the List Data Structures:

You are planning a transit network line (TNL) from Point A to Point B. You will allow 3 modes of transportation: train, bus, and Rickshaw. You can imagine that there is a number line from Point A to Point B, where Point A represents Number 0 and Point B represents Number K-1. In other words, the location of Point A is 0 and the location of Point B is K-1. So, there are a total of K spots, i.e., rickshaw stops (RS) where a rickshaw can stop (and hence take/leave a passenger). You can only move forward along this number line in a journey. The train is much faster than the bus, and the bus is much faster than the rickshaw. Unfortunately, the train can only stop at points where a train station (TS) exists, and the bus can only stop at points where a bus stop (BS) exists. A bus stop exists at every train station. Please see Table 2 for an example illustration.

Table 2: This table illustrates the idea of the TNL from Points A to B. Point A represents 0 and Point B represents 14. So there are a total of 15 rickshaw stops. RS, BS and TS mean rickshaw stop, bus stop and train station, respectively. In this scenario, there are 7 bus stops and 4 train stations.

A														B
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS
BS		BS			BS	BS			BS		BS			BS
TS					TS						TS			TS

You need to implement the above-described TNL leveraging your list implementations. You need to write a program that takes input information (format specified below) and is able to do various tasks as described below. You must write just one piece of code that will work on both Arr and LL implementations, the only difference between the two pieces of code will be the list object instantiations. So, you can only use the methods listed in Table 1.

Task 1:

Output an encoding of the network (format specified below).

Further tasks may be provided during the sessional as online tasks. Further instructions if needed will be provided then.

Input format:

One line containing the number of rickshaw stops: K.

One line containing the number of bus stops: L.

One line containing the location of each bus stop, space separated: B_0, B_1, … ,B_{L-1}

One line containing the number of train stations, M.

One line containing the location of each train station, space separated: T_0, T_1, … ,T_{M-1}

One line containing the Task number to perform: T

Example input corresponding to Table 2 and for Task 1:

15

7

0 2 5 6 9 11 14

4

0 5 11 14

1

<u>Task 1 Output Format:</u>

Output will in spirit follow the format of Table 2 as follows.

First line will print the numbers 0 to K-1, comma separated.

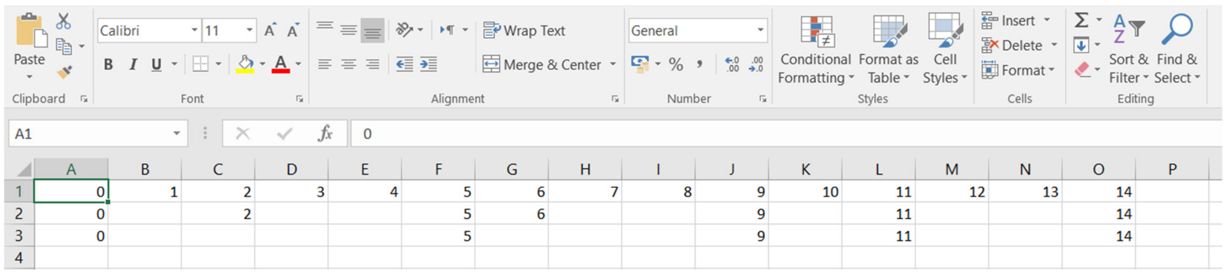
Second line will print the location of the bus stops appropriately comma separated so that they are aligned with the corresponding location in the first line

Third line will print the location of the train stations appropriately comma separated so that they are aligned with the corresponding location in the first line and second line.

Example output corresponding to Table 2 and Task 1 is as follows:

```
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14
0,,2,,,5,6,,,9,,11,,,14
0,,,,,5,,,,9,,11,,,14
```

Equivalently, if the above is saved as a csv file and then opened, we will get the following in excel (Figure 1):



0

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
2	0		2			5	6			9		11			14	
3	0					5				9		11			14	
4																

Figure 1: csv file corresponding to example output above

Submission Guidelines:

1. **Create a directory with your 7-digit student id as its name**
2. **You need to create separate files for the Arr implementation code (e.g. Arr.cpp/Arr.py), LL implementation code (e.g. LL.cpp/LL.py) putting common codes in another file. Create a separate file for the main function implementing the TNL (e.g., TNL.cpp/TNL.py). Your TNL implementation must**

import/include your array and LL implementations as header files, and you must use your own array/LL implementations for the TNL task. No other built in data structure can be used.

3. **Put all the source files only into the directory created in step 1. Also create a readme.txt file briefly explaining the main purpose of the source files.**
4. **Zip the directory (compress in .zip format. Any other format like .rar, .7z etc. is not acceptable)**
5. **Upload the .zip file on Moodle in the designated assignment submission link. For example, if your student id is 1905xxx, create a directory named 1905xxx. Put only your source files (.c, .cpp, .java, .h, etc.) into 1905xxx. Compress the directory 1905xxx into 1905xxx.zip and upload the 1905xxx.zip on Moodle.**

Failure to follow the above-mentioned submission guideline may result in upto 10% penalty.

Submission Deadline: 26/11/2021 11 PM.

Evaluation Policy:

Sl.	Item	
1	Array based Implementation	25%
2	List Based Implementation	25%
3	Support of various types of 'elements'	15%
4	TNL Implementation	35%