Assignment 1: List

Task 1:

You need to implement a List ADT (abstract data type) as defined in Table 1. Assume that the elements of the List are integers.

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	insert(item)	19		<20, 23 | 19, 12, 15>	Inserts an element at the current location.
2	remove()	-	12	<20, 23 | 15>	Remove and return the current element.
3	moveToStart()	-		< | 20, 23, 12, 15>	Set the current position to the start of the list
4	moveToEnd()	-		<20, 23, 12 |15>	Set the current position to the end of the list
5	prev()	-		<20 | 23, 12, 15>	Move the current position one step left. No change if already at the beginning.
6	next()	-		<20, 23 , 12 | 15>	Move the current position one step right. No change if already at the end.
7	length()	-	4	<20, 23 | 12, 15>	Return the number of elements in the list.
8	currPos()	-	2	<20, 23 | 12, 15>	Return the position (in the list) of the current element.
9	moveToPos(int pos)	1		<20 | 23, 12, 15>	Set current position.
10	getValue()	-	12	<20, 23 | 12, 15>	Return the current element.

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.

You should have an initialization function (e.g., init(<parameter(s)>)) which should set up the list in both implementations. You can decide which parametrs should be passed to this function. For example, for the Arr implementation, the init() function will need to be used to allocate the memory for the underlying array. You may implement extra helper functions but those will not be available for programmers (i.e., users of your data structure) to use. So, while using the list implementations, one can only use the methods listed in Table 1 and the init() function for initialization. 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 List 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<=10 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.

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 "|" as defined/shown in Table 1. Then for each task, it will output two lines as follows:

the first line will output the items of the list, space separated identifying the current position using "|" as defined/shown in Table 1 and the second line will output the return value (if available) or -1 (if no return value is expected).

Using the List Data Structures:

Task 2:

You need to implement the following functions as a user of the data structures. So, to implement these you only have the functions of Table 1 available to you.

Table 2

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	append(item)	19		<20, 23 | 12, 15,19>	Appends an element at the end of the list.
3	Search(item)	23	1	<20, 23 | 12, 15>	Search returns the position of the element item or -1 if not found.
4	Search(item)	29	-1	<20, 23 | 12, 15>	

Input/output format for Task 2 will be identical to Task 1.

Task 3:

You have to manage a car rental system (CRS) as follows. You have X cars (each car is labelled from 1 to X) and Y garages (each garage is labelled from 1 to Y). Each garage can park Z number of cars. At any point of time, some cars are in rental and some cars are waiting in the garages.

Now when a request for a rental car comes, you assign a car from a non-empty garage. However, the rule is that (a) if more than one garages are non-empty, you need to select the garage having the least label and if k>0 cars are parked in the selected garage, you will have to assign the car having the least label. If no cars are available overall (i.e., all garages are empty), you need to give a message that no cars are available.

Now when a car returns, you park it to a non-full garage (i.e., a garage having <Z cars parked). However, the rule is that (a) if more than one garages are non-empty, you need to select the garage having the highest label and (b) you

will have to park the car after the car having the highest label in the selected garage (in case of an empty garage just park it in that garage). If all garages are full, you need to give a message accordingly.

You need to implement the above-described CRS 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 variable declaration. So, you can only use the methods listed in Table 1 and the init() function.

Input format:

One line containing the number of cars: X.

One line containing the number of garages: Y.

One line containing the maximum number of parking spots in a garage: Z.

Y lines each containing the list of cars parked in each garage as follows. This like will have space separated p+1 numbers, $g\ C_{x_1}, C_{x_2}, \ldots, C_{x_p}$, such that C_{x_i}, $1 \leq x_i \leq p$ represents the p cars parked at garage g $(1 \leq g \leq Y)$.

Then we may have multiple lines, each line containing one of the two events 'req' (requesting a car) or "ret p" (returning car p). An "end" in a line (instead of one of the two events) will mean the end of the input and the program will terminate.

Output Format:

The output will provide the current status of the garage in a sorted manner, i.e., in the first line garage 1, then garage 2 and so on. Each such line will output space separated labels of the cars as they are parked in the garage. If no cars are parked in a garage, an empty line must be printed.

The output will start with the initial status of the garages, i.e., Y lines depicting the initial status. Then for each event, the status after the event must be output. So if there are a total of 6 events in the input, 1 + 6 status' will have to printed as mentioned above.

Example input:

```
10
4
4
2       3       10      7
1       1       2       4       8
```

```
3
4 6 5 9
req
req
req
req
ret 8
ret 1
end
```

Example Output:

```
1 2 4 8
3 10 7

6 5 9
2 4 8
3 10 7

6 5 9
4 8
3 10 7

6 5 9
8
3 10 7

6 5 9

3 10 7

6 5 9

3 10 7
```

6 5 9 8

3 10 7
1
6 5 9 8

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.c), LL implementation code (e.g. LL.cpp/LL.c) putting common codes in another file. Create a separate file for the main function implementing the Task 2 and Task 3 (e.g., Task_x.cpp/Task_x.c). These must import/include your array and LL implementations as header files, and you must use your own array/LL implementations for these tasks. 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 2105xxx, create a directory named 2105xxx. Put only your source files (.c, .cpp, .h, etc.) into 2105xxx. Compress the directory 2105xxx into 2105xxx.zip and upload the 2105xxx.zip on Moodle.**

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

Submission Deadline: 17/06/2023 11:59 PM. The deadline will not be extended beyond this as otherwise you will have to stay awake and that will affect your class performance in the Sunday classes.

Evaluation Policy:

Sl.	Item	
1	Task 1	

A	Array based Implementation	30 %
B	List Based Implementation	35 %
2	Task 2	15
3	Task 3: CRS Implementation	20 %

Rubrics for Task1 (for both implementations):

Per function marks	5
Total (10 functions)	5 x 10 = 50

Rubrics for Task 2

Per function marks	5
Total (4 functions)	5 x 4 = 20

For each function in Tasks 1 and Tasks 2:

Grade	Definition	Marks
D	Just attempted to do this	2
C	Some functionality	3
B	Major functionality	4
A	Full functionality	5

Rubrics for Task 3

A	Creating the required lists	10
B	Handling req event	10
C	Handling ret event	10
D	I/O formatting	5

	Total	35

For each 3.A – 3.C

Grade	Definition	Marks
D	Just attempted to do this	2
C	Some functionality	4
B	Major functionality	8
A	Full functionality	10

For 3.D just give marks directly.