Bangladesh University of Engineering and Technology

January 2024, CSE 106

Assignment 1

Array list and Linked list

In this assignment, you will implement the List abstract data type

Abstract data types are data types that are defined by the set of operations. These operations dictate their behavior. The user should only know “What can I do with this data type?”. It is not necessary for the user to learn “How is the data structure doing all these operations?”. But as a programmer concerned with creating the ADT, we have to be able to answer both questions.

A List is a data structure that holds a collection of elements or data as a list. For example, they can be a list of strings, list of integers, list of objects. It can support insertion of elements, deletion of elements, searching for an element, among other things.

Task: Implement the List ADT

For the first task, implement the List ADT. A List can hold any number of elements. For this offline, these *elements will be of integer data type*. The ADT will support several operations, which are demonstrated in table 1.

The table assumes that the list contains the values 20, 13, 5 and 7 (in that order), and currently the ADT is in the position of 2. We can show this information as below:

< 20 13 |5 7 >

Here, the list is enclosed by the symbols ‘<’ and ‘>’, the elements (integers) are shown in the order they appear in the list, and the current position (position 2) is shown as a vertical bar.

There are

a) One space after ‘<’ and one space before ‘>’

b) Two spaces in between two elements.

c) No space in between ‘|’ and the element in that position.

Table 1

Function Number	Function name	Params	Returns	After function execution	Comment
1	insert(int elem)	19	-	< 20 13 |19 5 7 >	Insert an element in the current position
2	remove_at_current()	-	5	< 20 13 |7 >	Remove the element situated at the current position
3	find(int elem)	5 12	2 -1	< 20 13 |5 7 > < 20 13 |5 7 >	If the element exists in the list, returns the first position it exists in. If not, then returns -1
4	move_to_start()	-	-	< |20 13 5 7 >	Moves the current position to the start
5	move_to_end()	-	-	< 20 13 5 |7 >	Moves the current position to the end
6	prev()	-	-	< 20 |13 5 7 >	Moves the current position one step left. No change already at the beginning
7	next()	-	-	< 20 13 5 |7 >	Moves the current position one step right. No change if already at the end
8	get_current_position()	-	2	< 20 13 |5 7 >	Returns the current position
9	move_to_pos(int pos)	0 1 2 3	-	< |20 13 5 7 > < 20 |13 5 7 > < 20 13 |5 7 > < 20 13 5 |7 >	Moves the current position to the given parameter. The parameter will always be valid (not less than 0 or greater than the size/length)
10	get_size()	-	4	< 20 13 |5 7 >	Returns the number of elements present
11	get_value()	-	5	< 20 13 |5 7 >	Returns the element at the current position

12	append(int elem)	42	-	< 20 13 |5 7 42 >	Inserts the parameter at the end of the list
13	clear()	-	-	< >	Clears the list, making it empty and size 0
0	Exit the program	-	-	-	If we get 0, the program will end. Make sure to free the allocated spaces before the program ends

The operations can either modify the List at the current position (insert, remove_at_current), modify the current position (prev, next, move_to_start, move_to_end, move_to_pos), get information from the current position (get_value,get_current_position), clear the list, find elements from the list, or append to the end of the list.

Do not modify the current position while modifying the list. For example, while appending an element, make sure that the current position stays where it was before appending.

You will implement List ADT using the

a) Array list implementation and

b) Linked list implementation

Array list:

An array list implementation will keep the elements in sequence, or in contiguous memory, just like an array. Since we cannot allocate unlimited memory for arrays, your implementation will have a certain capacity.

Let's say your array capacity is 10 initially. Meaning, you can hold up to 10 elements. Therefore, if you want to hold 11 or more elements in your List, you have to increase the array size. Once the 11th element is inserted, you will *double the size of the array,* making it 20, and then put the 11th element in its appropriate position. After that, if there is a 21st element to be inserted, you will again *double the capacity*, making it 40. This way you can give the user the illusion of unlimited memory.

You will also have to halve the capacity when the number of elements present in the list is less than 25% of the total capacity (e.g., when there are 9 elements in an array of capacity 40, shrink the capacity of the array to 20). This will help us not consume unused memory.

You can define your own resize function to achieve both the doubling and halving of the capacity.

Linked list:

A linked list will not carry the elements contiguously, but you will not have to worry about the size or capacity of your implementation. What you will need to do is define a struct, that can carry an element, and some pointers. Let's say if you call that struct "node", then the members of node will be an element, and pointer to other nodes. You can define two pointers called "prev" and "next", which will respectively point to the previous node and next node. The first node (or head) will have Null as the previous node, and the last node (or tail) will have Null as the next node.

The most important thing in Linked list is to take note of the order of the update while doing any operation. Update the 'next' and 'prev' pointers so that no information is lost. For example, while inserting, make sure to update the 'prev' and 'next' pointers of the newly created node before updating the pointers of the nodes already in the list.

We will provide a skeleton code written in C for this assignment. Please go through it and fill in the unimplemented functions in the .h files for the code to work. The skeleton code is already configured to take the input as described below in the input format section. You should also modify the main function so that it works as described below in the output format section.

Read the comments in the skeleton code. It will help you figure out what you should do.

Input format:

The first line will contain two numbers ***N and C***, where *N* is the **initial length of the list** and *C* is the **total capacity**. It is ensured that $N < C$.

For the Array list implementation, *C* will be used. It will be ignored for the Linked list implementation. *Initially, the current position will be 0 for both implementations.*

The next line will have ***N integers***, the elements of the list.

From the 3rd line on, each line will contain two space separated numbers. A ***function number F,*** (from table 1) and a ***parameter, P***.

Call the function corresponding to the number *F,* using the parameter *P.* If the function does not use any parameter, ignore *P.* (e.g., if we get 1 10, we will insert 10 into the current position. Do this until *F=0,* which indicates that the program should terminate.

Output format:

Firstly, output the list enclosed by angular brackets ('<' and '>') with a vertical bar ('|') at the current position. Then print an empty line.

After each operation done on the list from the third line onwards, first you have to print a log of the operation done by the numbers *F* and *P*.

for example, for the list

< |20 13 5 7 >

Input line	Output lines
9 2	Moved the current position to 2 < 20 13 |5 7 >
1 10	Inserted 10 on position 2. < 20 13 |10 5 7 >
3 7	Found 7 in the list at position 4 < 20 13 |10 5 7 >
8 0	Current position is 2 < 20 13 |10 5 7 >

You can use appropriate log messages as per your wish. Make sure they convey how the array was modified or what operation was done properly.

You should also print a log message when you are using an Array List implementation and the array gets resized. Make sure to mention the previous capacity along with the new capacity.

Marks Distribution:

Topic		**Percentage**
Array List resize		5%
Array List implementation		45%
Initialization and clearing	5%	
Insert, Remove, Append, Search	20%	

Other functions	20%	
Linked List implementation		50%
Insert, Append, Remove, Find	25%	
Other functions	25%	
Total		100%

Submission Guidelines:

1) Create a new folder with your 7-digit student ID.

2) Copy your source files into the folder created in step 1. Your files must follow the structure mentioned in the requirements.

3) Zip the folder (along with the source files). Make sure it has the ".zip" extension. No other extension will be accepted.

4) Upload the zip file into the designated submission link on Moodle.

Suppose your student ID is 2205xxx. If so, create a folder named "2205xxx", put your source files in that folder, compress/zip the folder into 2205xxx.zip, and finally, upload 2205xxx.zip to Moodle.

Please make sure to follow the guidelines. *You might be penalized up to 10% of the total marks if you do not comply*. Also make sure to upload the correct zip file (the file is not corrupted or is not the solution to a previous assignment you did).

Submission Deadline:

September 15, 2024, 11:55 PM

Version History:

The document may be updated periodically to address any ambiguities, clarify instructions, or incorporate feedback from students to ensure the assignment is fully understandable and free of confusion. All changes in the document will be marked with red underline, along with a version change. Please keep yourself updated on the changes in the document.

Version 1.0

initial release