Homework 1: Playing with ArrayList

You will be given with a dynamic $ArrayList$ implementation in your class. Extend the implementation to include the following features:

Task 1: Add getLength function. Add a new function $getLength$ to the list. This function will return the current length, i.e., the number of items, of the list.

Task 2: Add insertItemAt function. Add a new function $insertItemAt$ to the list. This function will insert a new item at a given position. The prototype of the function is as follows: int insertItemAt(int pos, int item). The function will insert the given $item$ at the given pos. You have to do this by shifting only one item of the existing list. Note that the pos value can be larger or equal to $length$ variable. In such a case, you should not insert anything in the list. You may also require allocating new memory similar to $insertItem$ function if list is already full.

Task 3: Add shrink feature to the list. The current implementation expands its memory when it is full. This is done in the $insertItem$ function. When it finds that the $length$ variable has reached the value of $listMaxSize$, it allocates a new memory whose size is double the existing memory. Your job is to add ***shrink*** feature to the list. This feature will allow the list to ***shrink*** its memory whenever the $length$ variable has reached to half of the current size. In that case, you will reallocate a new memory whose size will be half the current size. However, if the current size of the list is LIST_INIT_SIZE, then you will not need to shrink the memory. Write a function $shrink$ that will shrink the list appropriately.

Task 4: Add deleteLast function. Add a new function $deleteLast$ to the list. This function will behave similar to existing $deleteItem$ function except that it will delete the last item of the list. You will not need to move any other items. So, this function will not have any input parameter. You must call the $shrink$ function after deletion. You will also require adding $shrink$ function calls inside $deleteItem$ and $deleteItemAt$ functions.

Task 5: Add clear function. Add a new function $clear$ to the list. This function will delete all items from the list and will de-allocate its memory. When a new item will be inserted next, the list must be allocated a new memory of size $LIST\_INIT\_SIZE$ again. You must write required codes in the $insertItem$ function to enable this.

Task 6: Add deleteAll function. Add a new function $deleteAll$ to the list. This function will delete all items from the list, but will not de-allocate its memory. However, it will shrink its memory to $LIST\_INIT\_SIZE$ if the list is currently consuming a memory whose size is larger than $LIST\_INIT\_SIZE$.

Task 7: Solve the balanced parenthesis problem. In this task, you will solve the balanced parenthesis problem using your list. In this problem, a parenthesis structure will be given as input, for example “(([]))”. You may store this in a ***string*** variable. Your program will determine whether the input parenthesis is structurally correct, i.e., it forms a well balanced parenthesis. The algorithm to check this is as follows:

1. Initialize your list at the beginning of your program.

2. Take input string from user.
3. Scan the characters of the input string one by one starting from the leftmost position. For each character, do any one of the following:
    a. If it is a "(", "{", or "[", then insert the character to the list by using $insertItem$ function. Do not worry about inserting characters to int type list. It will work perfectly!
    b. If it is a ")", "}", or "]", then do not insert it. First, check whether the list is empty. If it is, then this indicates an unbalanced parenthesis structure. If not empty, then delete one from the list by calling the $deleteLast$ function. Then check whether the removed item is the left matching of the currently scanned character. If it is, then carry on scanning. If not, then it denotes an unbalanced parenthesis structure. You may stop scanning at this point and go to Step 5 directly.
4. After scanning of all characters, check the length of the list by calling $getLength$ function. If it is not 0, then it implies that the input parenthesis was not balanced.
5. Give your output as "Balanced" or "Not Balanced".
6. Clear your list by calling the $clear$ function.

Sample input and output for Task 7 are given in the following table.

Input	**Output**
([])	Balanced
[(])	Not Balanced
(()[])	Balanced
(()[]	Not balanced

You must also satisfy the following requirements:

- You must extend the given code.
- You cannot use any advanced features of C library.
- You cannot use object oriented programming.
- You must *free* unused memory where it is required.
- For task 7, you may assume that the input will not contain any invalid characters.
- You must write two main functions. The first main function will contain test codes for Tasks 1-6. The other main function will contain test codes for Task 7. You will comment out one while you will demonstrate the other during evaluation.
- For any clarification and help, contact your teacher. I will be available in my room, #209. If I am not there, give me a call at my number 01674 069126.
- You may be provided an updated and corrected version of this document later if It is required.
- *You must not use other's code. You must not share your code. You must not copy from any other sources such as web, friends, relatives, etc. In all cases, you will earn a 0 and will move closer to getting an "F" grade in the course.*

Assignment designed by Sukarna Barua, *Assistant Professor, CSE, BUET*