Xiao Feng

Lab1 Analysis:

Data Structure:

I utilized stack data structure and implemented with Array. Stack is a reasonable choice to solve this problem because of stack's LIFO nature. It has the following benefits specifically for this lab problem:

- Stack allows comparing count of letters. To validate language1, I need to count the number of As and Bs in the string. I can utilize two stacks to store As and Bs separately while reading in string. After reading completed, two stacks pop at the same time until one stack is empty, then check if the other stack is also empty. If yes, then number of As and Bs is equal.
- Stack records sequence of each letter. To validate language 4, I used a stack to store all letters from input. Letters are not in their original order, but they are still saved in the same sequence, it is still the same letters next to each other. Therefore, I compare the sequence of A^nB^m with the stack repeatedly until un-match happens or stack is empty.
- Stack reverse sequence of the string. To validate my language5 (symmetric letters), I used a stack to store first half of letters, then compare stack with the last half of letters. In this way, I don't have to worry about reverse first half to compare with last half; stack did it for me because of its LIFO nature.

I choose Array to implement stack for couple reasons. First, input size is countable. While reading the input string, the size is determined, it is very reasonable to set the limit size for stack. Second, only character value type is stored on the stack, which meets homogeneous requirement. Also, array allows random access. In my design, I didn't utilize random access feature, but it offers some flexibility maybe applicable for some additional language type.

The complexity of each language checker:

L1: O(n) – it iterates through input string, pushes on two stacks and compares stacks by doing pop at the same time. Check if stacks are empty to determine language1 qualification.
Space usage: two arrays with size n

L2: O(n) – it iterates through input string, pushes all A's(before the first B appears) on one stack, pops while B was kicking in. Check if stack is empty to determine language2 qualification
Space usage: one array with size n

L3: O(n) – it iterates through input string, pushes A's(before the first B appears) twice on one stack, pops while B was kicking in. Check if stack is empty to determine language3 qualification
Space usage: one array with size n

L4: O(n) - it iterates through input string, pushes all characters on one stack. Save the very fist A^nB^m from very left in an array as base. Compare stack and base by comparison stack top and array from right to left. Pop stack if match, stop if any mismatch happens. Check if stack is empty to determine language4 qualification
Space usage: one array with size n, one array with size n/p

Xiao Feng

L5: O(n) - it iterates through input string, pushes first half of characters on one stack, pops if next character match with top of stack. Check if stack is empty to determine language5 qualification
Space usage: one array with size n

L6: O(n) - it iterates through input string if only contains As and Bs and first character is A, pushes first half of characters on one stack. Pops if next character matches with top of stack. Check if stack is empty to determine language6 qualification
Space usage: one array with size n

A recursive implementation can be applied to Language4 = (A^nB^m)^p. (A^nB^m)^p can be interpreted as (A^nB^m)*(A^nB^m)^p-1, which also equals to
(A^nB^m)*(A^nB^m)*)*(A^nB^m)^p-2. With stopping point equals 1, the recursive function is L4(p) = (A^nB^m)*L4(p). I think the recursive implementation will be better than the current one because I used two loops to check both formats within parenthesis and repeating overtimes. It is more intuitive and straight using a recursive function as discussed above.

Enhancement:
1. A language 6 to check if input qualify for A^nb^2nA^n
2. If an empty string appears, the output will interpret as "input contain 0 As and 0 Bs" which qualifies for language2(n=0), language3(n=0), language4(n=0, m=0, p=0), language6(n=0)
3. If a leading and/or trailing space exists. Program treat as typo and remove the leading and/or trailing
4. The program will treat lower case and upper case as the same letter from input. For example AAABBB equals aaABBB
5. Any space in between a string input with being removed. Example: "aa bb" equals to AABB

Learnings:
Lab1 is a good practice to implement a stack data structure. By creating the stack class from scratch, I had a better understanding of what are the basic functions for a stack, how each function is constructed and how to use them in completing a task. The development of language comparison process helps me get experience on using a stack structure. In the later development work, I'm more confident in applying stack and be more appreciated for the stack library Java already have.

Something I'd do differently next time is to implement list in stack. The benefit of list is dynamic allocation. While array requires a fixed size, I tend to assign the max size (in this case, max size is numbers of input character) to every stack to avoid overflow. It is not as space efficient as a list implementation.