資料結構與進階程式設計（108-2）

手寫作業七

B08705034 資管一 施芊羽

Qusetion 1

Answer:

```
Display(aStack: Stack):
    Initialize newStack as an empty stack
    while aStack.isEmpty() is false:
        newStack.push(aStack.peek())
        aStack.pop()
    while newStack.isEmpty() is false
        newChar = newStack.peek()
        newStack.pop()
        Write newChar
```

My solution will have a input data `aStack` , and I'll create a `newStack` to store the the indices in `aStack` from the back. Therefore, after all data in `aStack` have been stored in `newStack` , we can write each item in the stack from keeping `peek` and `pop` . And the output result will exactly display the `aStack` from its beginning.

Question 2

Answer:

```
bool Check(input: string):
    Initialize aStack is an empty stack
    for all characters in input:
        if the character is a bracket:
            if the character is a open bracket:
                aStack.push(the character)
            else if the character is a close bracket:
                if aStack.peek() is pair to the character:
                    aStack.pop()
                else:
                    return false
        move to the next character

    return true
```

My solution sets a `input` (type : `string`) to the function and applys the way we create the basic functions of `Stack` . I'll check if the current character of `input` is a bracket or not, if it's a open bracket(no matter which type of brackets it is), push it to the `aStack` ; if it's a close bracket, do `aStack.peek` to check if there's a open bracket pair to the current bracket, if so, `pop` that paired open bracket or return `false` if not.

Question 3

Answer:

The $BigO$ of the simple calculator function in the slides will be $O(n)$.

The $BigO$ of `Stage1` will be $O(n)$. Suppose that there'll be $\frac{n+1}{2}$ tokens that are operators and $\frac{n-1}{2}$ will be operands that will be put into the `Stack1` . Consider the times the fuction that will run through the `while` -loop, the loop will run at most $\frac{n-1}{2}$ times, and the `if-else` condition, will run n times in total. Therefore, the $BigO$ notation of the `Stage1` can be written as $O(n)$.

The $BigO$ of `Stage2` will be $O(n)$. Suppose that all the operands and operators remain in the two stacks, it'll take at most n times to run the the `while` loop in `Stage2` . Therefore, the $BigO$ of the `Stage2` can be written as $O(n)$.