# **Introduction**

The challenge involves simulating an old phone keypad where each button corresponds to multiple letters. Pressing a button multiple times cycles through its letters, and a pause (represented by a space) is required to type two characters from the same button consecutively. Additionally, the keypad includes a backspace key (\*) and a send key (#), which signifies the end of the input.

### **Keypad Mapping**

Here’s the mapping of the phone keypad:

* **2**: ABC
* **3**: DEF
* **4**: GHI
* **5**: JKL
* **6**: MNO
* **7**: PQRS
* **8**: TUV
* **9**: WXYZ
* **0**: Space
* **\***: Backspace (removes the last character)
* **#**: Send (indicates the end of the input)

### **Rules**

1. **Character Mapping**: Each button press corresponds to a letter. For example, pressing 2 once gives A, twice gives B, and three times gives C. Pressing 2 four times cycles back to A.
2. **Pause Handling**: To type two characters from the same button consecutively, you must pause (represented by a space). For example, 222 2 22 translates to CAB.
3. **Backspace**: The \* key removes the last character from the output.
4. **Send**: The # key indicates the end of the input and should not be included in the output.

### **Example Inputs & Expected Outputs**

| **Input** | **Output** | **Explanation** |
| --- | --- | --- |
| "33#" | "E" | 3 twice → E |
| "227\*#" | "B" | 22 → C, \* removes it, 2 → B |
| "4433555 555666#" | "HELLO" | Correct letter selection with pause |
| "8 88777444666\*664#" | “TURING” | Complex case |

### 

### 

### **Approach**

### **Input Parsing:** Iterate through the input string, processing each character.

### **Character Mapping:** Used a dictionary to map each number to its corresponding letters.

### **Counting Presses:** Count consecutive presses of the same button to determine the correct letter.

### **Pause Handling:** Use spaces to reset the count for the same button.

### **Backspace Handling:** When encountering \*, remove the last character from the output.

### **Send Handling:** When encountering #, stop processing and return the output.

## **Conclusion & Key Takeaways**

* **Performance Optimization**: Uses StringBuilder for efficient string manipulation.
* **Error Handling**: Handles backspace, invalid inputs, and empty cases gracefully.
* **Code Modularity**: The function is structured for easy extensibility.
* **Test Coverage**: Unit tests validate correct behavior in various edge cases.