JavaScript Baseball Game

Challenge

You are keeping the scores for a baseball game with strange rules. At the beginning of the game, you start with an empty record.

You are given a list of strings `operations`, where `operations[i]` is the i^{th} operation you must apply to the record and is one of the following:

- An integer `x`.
- Record a new score of `x`.
- `'+'` records a new score that is the sum of the previous two scores.
- `'D'` records a new score that is the double of the previous score.
- `'C'` invalidates the previous score, removing it from the record.

Return the sum of all the scores on the record after applying all the operations.

The test cases are generated such that the answer and all intermediate calculations fit in a 32-bit integer and that all operations are valid.

1st Example

```
Input: ops = ['5','2','C','D','+']
Output: 30
Explanation: '5' - Add 5 to the record, record is now [5].
             '2' - Add 2 to the record, record is now [5, 2].
             'C' - Invalidate and remove the previous score,
                   record is now [5].
             'D' - Add 2 * 5 = 10 to the record,
                   record is now [5, 10].
             '+' - Add 5 + 10 = 15 to the record,
                   record is now [5, 10, 15].
            The total sum is 5 + 10 + 15 = 30.
```

2nd Example

```
Input: ops = ['5','-2','4','C','D','9','+','+']
Output: 27
Explanation: '5'  - Add 5 to the record, record is now [5].
             '-2' - Add -2 to the record,
                    record is now [5, -2].
             '4'  - Add 4 to the record,
                    record is now [5, -2, 4].
             'C'  - Invalidate and remove the previous score,
                    record is now [5, -2].
             'D'  - Add 2 * -2 = -4 to the record,
                    record is now [5, -2, -4].
             '9'  - Add 9 to the record,
                    record is now [5, -2, -4, 9].
             '+'  - Add -4 + 9 = 5 to the record,
                    record is now [5, -2, -4, 9, 5].
             '+'  - Add 9 + 5 = 14 to the record,
                    record is now [5, -2, -4, 9, 5, 14].
            The total sum is 5 + -2 + -4 + 9 + 5 + 14 = 27.
```

3rd Example

```
Input: ops = ['1','C']
Output: 0
Explanation: '1' - Add 1 to the record,
                   record is now [1].
             'C' - Invalidate and remove the previous score,
                   record is now [].
             Since the record is empty, the total sum is 0.
```

Constraints

- `1 <= operations.length <= 1000`
- `operations[i]` is `'C'`, `'D'`, `'+'`, or a string representing an integer in the range `[-3 * 10⁴, 3 * 10⁴]`.
- For operation `'+'`, there will always be at least two previous scores on the record.
- For operations `'C'` and `'D'`, there will always be at least one previous score on the record.

Solution

```
const calPoints = (ops) => {
    const stack = [];
```

Solution continues on next page...

```
    for (let i = 0; i<ops.length; i++) {
            switch(ops[i]) {
            case 'D': {
                const last = stack[stack.length - 1];
                stack.push(last * 2);
                break;
            }
            case 'C': {
                stack.pop();
                break;
            }
            case '+': {
                const firstScore = stack[stack.length - 2];
                const secondScore = stack[stack.length - 1];
                stack.push(firstScore + secondScore);
                break;
            }
            default: {
                stack.push(Number(ops[i]));
                break;
            }
        }
    }

    return stack.reduce((prev, cur) => prev + cur, 0);
};
```

Explanation

I've defined a function called `calPoints` that takes an array of strings, `ops`, as input. The purpose of this function is to perform a series of operations on a stack and return the sum of all the values in the stack.

Inside the function, an empty array called `stack` is created to store the values.

A `for` loop is used to iterate through each element in the `ops` array. Within the loop, a `switch` statement is used to handle different cases based on the value of `ops[i]`.

If `ops[i]` is equal to `'D'`, the code inside the `case 'D'` block is executed. It retrieves the last element from the `stack` using `stack.length - 1`, multiplies it by `2`, and pushes the result back into the `stack`.

If `ops[i]` is equal to `'C'`, the code inside the `case 'C'` block is executed. It removes the last element from the `stack` using the `pop()` method.

If `ops[i]` is equal to `'+'`, the code inside the `case '+'` block is executed. It retrieves the second last element from the `stack` using `stack.length - 2`, retrieves the last element using `stack.length - 1`, adds them together, and pushes the result back into the `stack`.

If none of the above cases match, the code inside the `default` block is executed. It converts the string `ops[i]` to a number using `Number(ops[i])` and pushes it into the `stack`.

After the loop finishes, the `stack` is returned.

The `reduce` method is called on the `stack` array to calculate the sum of all the values. The initial value is set to `0`.

The final sum is returned as the result of the `calPoints` function.

In summary, this function performs various operations (double,

remove, add, or push a number) on a stack based on the values in the `ops` array. It maintains the stack and calculates the sum of all the values in the stack, which is then returned as the output of the function.

Author: Trevor Morin