Submission:

Psid: 2131201
Sumbission-id: fe53ca11-2298-4894-950e-8627afe026a5 (picture of the submission attached at the end of the document)

A Brief Description:

Designing a solution to this problem leads to a requirement of three methods:

1. isValid: this method takes in three strings (the first two strings that are to be added and the third string containing the coded sum of these two strings), the complete list of digit mappings and returns True if the digit mappings are valid and false otherwise.
2. notFound: This method takes in the list of mappings (may be incomplete) and a value and returns True if said Value is found in mappings list, and False otherwise. This method is a helper method to avoid reassigning digits that have already been assigned in the backtrack function.
3. Backtrack: This is the function where all the recursive magic happens. The function takes in a list of inputs:
    a. An index to keep track of the symbol in exploration.
    b. The first string input containing the list of symbols
    c. The three strings that are used to check the validity of the digits.
    d. The list of digit mappings for the string of symbols (can be incomplete)

    Base case:
    - if the symbol Index is at the end of the list of symbols, run isValid on the mappings and return the results,

    Recursive Case:
    - iterate over every unused digit in 0 to 9 for current symbol
    - For each unused digit, assign it to the current symbol, and then recursively call the function on the next digit.
    - If the function call returns True, return True (return the valid result up the chain)
    - Otherwise reset the current symbol to -1
4. Main function: here everything is executed:
    a. Input the four strings in the following order: symbollist, str1, str2, str3; initialize a list to store the digit mappings.
    b. Reverse the three strings str1 thru 3 to make it easier to validate in isValid.
    c. If the backtrack function with initial conditions returns True, then print out the list of digit assignments in mappings. End main function.
    d. Otherwise output no valid mapping was found.

Pseudocode:

```
bool notFound(mappings[], int value):
    for mapping in mappings:
        if mapping == value:
            return False   //*returns false if the value IS INDEED found in mappings
    return False //*the value is good to go.

bool isValid(mappings, str1, tr2, str3):
    int carry = 0
    for i in range str3.length:
        grab the digit assignments for letters at i in str1 and str2
        if theres one more digit in str3, then set str2 and str1's digits to 0 //*this takes care of carry overflow.
        if (digit1 + digit2 + carry) %10 != digit3:
            return false //*the assignment is invalid.
        carry = floor((digit1 + digit2)/10) //*or do a simple boolean check. wtvr works tbh.
    return true;

bool backtrack(int symbolindex, mappings[], string str1, string str2, string str3, string symbollist)
    if(symbolindex is at symbollist's end)
        reutrn isValid(mappings, str1, str2, str3) //*base case check if all mappings are correct
    for every digit at the index of symbollist:
        if(notFound(mappings, i)):
            mappings[symbolindex] = digit //*set that digit to that symbol cuz it hasn't been asigned yet.
            if(backtrack(symbolindex++, mappings, str1, str2, str3, symbollist)) //*if you found the correct one return it
                return True
            else
                mappings[symbolindex] = -1 //*reset the mappings.
    return false; //*catchall for oopsies; also the function requires a return to avoid seg faults and race conditions

main:
    input >> symbollist
    input >> str1
    input >> str2
    input >> str3
    reverse str1 to 3
    initailize mappings[]
    if(backtracking(0, mappings[], str1, str2, str3, symbollist))
        for mapping in mappins:
            print mapping
        return 0;
    cout no valid mapping found
    return 0;
*/
```

Justification of Correctness:

The backtracking algorithm explores all possible assignments of digits to symbols. At each step, it assigns an unused digit to a symbol and recursively continues the search. If a valid assignment is found, it returns the assignment. If not, it backtracks and tries a different assignment until all possibilities are exhausted.
The algorithm's correctness is guaranteed because it systematically explores all possible assignments and checks if they satisfy the equation. If a valid assignment exists, the algorithm will find it through the recursive search.

Justification of Runtime Complexity:

The backtracking algorithm explores all possible assignments for each symbol. In the worst case, it explores all permutations of the digits from 0 to 9 for each symbol. Therefore, the time complexity is O(10^n), where n is the number of symbols.

Compilation error msg:

takes 1.315897718s

---

test 0 PASS

test 1 PASS

test 2 FAIL

program output:

```
no valid mapping found
```

expected output:

```
704591682
```

test 3 FAIL

program output:

```
no valid mapping found
```

expected output:

```
91685304
```

test 4 FAIL

program output:

```
no valid mapping found
```

expected output:

```
84692501
```

test 5 FAIL

program output:

```
no valid mapping found
```

expected output:

```
712498053
```

test 6 FAIL

program output:

```
no valid mapping found
```

expected output:

```
7234159806
```

test 7 FAIL

program output:

```
no valid mapping found
```

expected output:

```
3819574206
```

test 8 FAIL

program output:

```
no valid mapping found
```

expected output:

```
7315468902
```

takes 50.813836ms