```c
#include <stdio.h>
#include <string.h>

typedef struct Token{
    char c;
    char rep;
}Token;

int checkRule(Token *pattern, char* message){
    int lenMessage = strlen(message);
    // i is an index into the rule string
    // j is an index into the pattern string
    int j = 0, i = 0;
    while(1){ // Read the entire rule string
        char c = pattern[i].c; // Character to expect in message
        char rep = pattern[i].rep; // How many times to expect it
        if(c < 0)
            break; // No more pattern left
        if(rep == '*'){ // Zero or more occurrences of c
            while(j < lenMessage && message[j] == c)
                j++; // Keep matching
        }else if(rep == '+'){ // One or more occurrences of c
            if(j >= lenMessage) // Incomplete message
                return 0;
            if(message[j] != c)
                return 0; // At least one match necessary
            while(j < lenMessage && message[j] == c)
                j++; // Keep matching
        }else{ // Repetition is a number
            int count = rep - '0';
            for(; count > 0; count--){
                if(j >= lenMessage) // Incomplete message
                    return 0;
                if(message[j] != c)
                    return 0; // Message failed to match the character
                j++;
            }
        }
        i++;
    }
    if(j < lenMessage) // Unmatched characters in the pattern
        return 0;
    return 1; // Nothing broke so we must have matched the whole rule
}

// Reduce the pattern sent and return the length of reduced pattern
void reducePattern(Token *old, Token *new){
    char curr = -1;
    // i is an index into the old pattern
    // j is an index into the new pattern
    // minRep stores how many times the current character must repeat
    // star is a flag storing if there is a variable repetition clause
    int minRep = 0, star = 0, i = 0, j = 0;
    while(1){
        // We have a new character, write the old compressed token
        if(old[i].c != curr){
            // However, dont do so if we are just starting
            // since the old token has a junk char in the beginnind
            if(i > 0){
                // Dont write a number token if minRep = 0
                if(minRep > 0){
                    new[j].c = curr;
                    new[j].rep = minRep + '0';
                    j++; // Advance the token
                }
                if(star){ // Do we have to print a * token?
                    new[j].c = curr;
                    new[j].rep = '*';
                    j++; // Advance the token
                }
            }
            // Reset things for a fresh start
            curr = old[i].c;
            star = 0;
            minRep = 0;
        }
        if(old[i].c < 0)
            break; // The old pattern is over
        switch(old[i].rep){
            case '*':
                star = 1;
                break;
            case '+':
                star = 1; // + does allow for variable repetition
                minRep++; // At least one occurrence is mandated
                break;
            default:
                minRep += old[i].rep - '0';
                break;
        }
        i++;
    }
    new[j].c = -1; // Delimiter
}

int main(){
    Token pattern[51], reduced[102];
    char message[100], tmp[100];
    gets(tmp); // Read in the pattern string
    // Store it in a nice structured format
    int i = 0, j = 0;
    for(; j < strlen(tmp); i++){
        pattern[i].c = tmp[j++];
        pattern[i].rep = tmp[j++];
    }
    pattern[i].c = -1; // Delimiter token
    gets(message);

    reducePattern(pattern, reduced);
    for(int i = 0; reduced[i].c > 0; i++)
        printf("%c%c", reduced[i].c, reduced[i].rep);
    printf("\n");

    if(checkRule(reduced, message))
      printf("YES");
    else
      printf("NO");
}
```