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

// Is the input string of given length a palindrome?
int isPalindrome(char *str, int len){
    for(int i = 0; i <= len - 1 - i; i++)
        if(str[i] != str[len - 1 - i])
            return 0;
    return 1;
}

// Can we make the input string a palindrome by adding pad-many
// characters to the right of the given string?
// Trick: a string can be made a palindrome by adding n characters
// to the right if and only if the string starting from the (n+1)-th
// character onward is already a palindrome. For example, "abede"
// can be made a palindrome by appending only 2 characters since
// the string 3rd character onward is ede which is a palindrome!
int makePalindromeAppend(char* str, int len, int pad){
    if(!isPalindrome(str + pad, len - pad))
        return 0;
    return 1;
}

// Can we make the input string a palindrome by adding pad-many
// characters to the left of the given string?
// Trick: a string of length k can be made a palindrome by adding 3
// characters to the left if and only if the first k-3 characters of
// the string already form a palindrome. For example, "edens"
// can be made a palindrome by prepending only 2 characters since
// the first 3 characters i.e. "ede" form a palindrome
int makePalindromePrepend(char* str, int len, int pad){
    if(!isPalindrome(str, len - pad))
        return 0;
    return 1;
}

int main(){
    char str[100];
    gets(str);
    int len = strlen(str), padding, padLeft, padRight;
    padding = 0;

    // Try making it a palindrome by padding to the right
    while(!makePalindromeAppend(str, len, padding++));
    padRight = padding - 1;

    // Try making it a palindrome by padding to the left
    padding = 0;
    while(!makePalindromePrepend(str, len, padding++));
    padLeft = padding - 1;

    // Did we require a shorter padding while prepending?
    if(padLeft < padRight){
        printf("%d\n", len + padLeft);
        for(int i = 0; i < padLeft; i++)
            printf("%c", str[len - 1 - i]);
        printf("%s", str);
    // or did we require a shorter padding while appending?
    }else if(padLeft > padRight){
        printf("%d\n%s", len + padRight, str);
        for(int i = padRight - 1; i >= 0; i--)
            printf("%c", str[i]);
    // If both required the same lenght, choose lexicographically
    // as asked in the question
    }else if(padLeft == padRight){
        int choosePadRight = 1; // Is it more advantageous to do a right pad?
        for(int i = 0; i <= len - 1 - i; i++){
            // If at any point I see an asymmetry I would know
            // whether to left pad or to right pad.
            if(str[i] < str[len - 1 - i]){
                choosePadRight = 1;
                break;
            }
            if(str[i] > str[len - 1 - i]){
                choosePadRight = 0;
                break;
            }
            // If str[i] == str[len - 1 - i] keep searching
        }
        if(choosePadRight){
            printf("%d\n%s", len + padRight, str);
            for(int i = padRight - 1; i >= 0; i--)
                printf("%c", str[i]);
        }else{
            printf("%d\n", len + padLeft);
            for(int i = 0; i < padLeft; i++)
                printf("%c", str[len - 1 - i]);
            printf("%s", str);
        }
    }
    return 0;
}
```