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

// Take a number and print it textually
// For non-positive numbers it returns the empty string
void itoa(int n, char *str){
    if(n <= 0){
        str[0] = '\0';
        return;
    }
    int i, len = floor(log10(n) + 1); // number of digits in n
    for(i = len - 1; i >= 0; i--){
        str[i] = '0' + (n % 10);
        n /= 10;
    }
    str[len] = '\0';
}

// n tell us which number is to be partitioned
// next tells us the location in the string where to print next
// min tell us the minimum number we must choose next
void partition(char *str, int n, int next, int min){
    if(n == 0){
        str[next] = '\0';
        printf("%s\n", str);
        return;
    }
    int i;
    // If this is not the first number in the partition
    // we need a plus sign before we print the next number
    if(next)
        str[next++] = '+';
    // Start from min so that numbers in a partition are always
    // in a non-decreasing order. This ensures that we will never
    // repeat a partition twice
    for(i = min; i <= n; i++){
        itoa(i, str + next);
        // We have already absorbed i so now partitions of n-i needed
        // All future numbers in this partition must be at least i
        partition(str, n - i, next + strlen(str + next), i);
    }
}

int main(){
    char str[1000];
    // This code will output partitions in lexicographically increasing
    // order. However, lexicographic order means something a bit different
    // for this code. Nevertheless, no partition will be repeated twice.

    // This code can handle any positive number, with any number of digits,
    // so long as writing the parition does not take more than 999 chars
    partition(str, 20, 0, 1);
    return 0;
}
```