```c
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */


char** fullJustify(char** words, int wordsSize, int maxWidth, int* returnSize) {
    int* stack;
    int stack_ptr;
    int stack_index;
    char** result;
    int cur_word_index;
    int word_index;
    int space_index;
    int per_space;
    int index;
    int length;
    int tmp_length;
    int str_len;
    int space;
    int space_remain;


    stack_ptr = -1;
    stack = (int*)malloc(sizeof(int)*wordsSize);
    result = (char**)malloc(sizeof(char*)*wordsSize);
    length = 0;
    word_index = 0;
    *returnSize = 0;

    for(index = 0; index <= wordsSize; index++)
    {
        str_len = (index == wordsSize) ? maxWidth : strlen(words[index]);
        tmp_length = length + str_len;


        if( (tmp_length + stack_ptr) < maxWidth)
        {
            stack_ptr++;
            stack[stack_ptr] = index;
            length = tmp_length;
        }else
        {
            index--;
            result[*returnSize] = (int*)malloc(sizeof(int) * (maxWidth+1) );

            space = maxWidth - length;


            cur_word_index = 0;
            stack_index = -1;

            while(stack_index < stack_ptr)
            {
                word_index = 0;
                space_index = 0;
                stack_index++;

                if(stack_ptr <= 0)
                {
                    per_space = space;
                    space_remain = 0;
                }else
                {
                    if(index < (wordsSize - 1))
                    {
                        per_space = (stack_ptr == stack_index) ? 0 : space / (stack_ptr - stack_index);
                        space_remain = (stack_ptr == stack_index) ? 0 : space % (stack_ptr - stack_index);
                        space_remain = (space_remain > 0) ? 1 : 0;
                    }else
                    {
                        per_space = (stack_ptr == stack_index) ? space : 1;
                        space_remain = 0;
                    }
                }



                while(words[stack[stack_index]][word_index] != '\0')
                {
                   result[*returnSize][cur_word_index] = words[stack[stack_index]][word_index];
                   word_index++;
                   cur_word_index++;
                }

                result[*returnSize][cur_word_index] = ' ';

                while(space && space_index < (per_space + space_remain) )
                {
                   result[*returnSize][cur_word_index] = ' ';
                   space_index++;
                   cur_word_index++;
                }

                space -= (per_space + space_remain);
            }

            //printf("%d, %d, %d\n", index, *returnSize, cur_word_index);
            result[*returnSize][cur_word_index] = '\0';

            (*returnSize)++;
            stack_ptr = -1;
            length = 0;
        }

    }

    return result;
}
```