11/03

T1　最大子数组和问题

题目

Description

小猫咪玩游戏，现在有n张牌依次放在地上，牌上有数字，数字的范围是-20~20，正数表示可以奖励对应数目的猫粮，负数表示减去对应数量猫粮。猫咪可以从中**连续**选择k张牌， 0≤k≤n（k=0表示不选牌，即0包猫粮），即能获得数量为k张牌的数字之和的猫粮。k的具体数值由猫咪决定，不作为输入。
实现一个函数，求出猫咪最多能获取几包猫粮，1≤数组长度≤100。已提供main函数解决输入输出，只需要实现函数即可。

Input

输入共n+1行，第一行为数组长度n，接下来的n行输入数组的数字。

Output

输出猫咪所能获得的最多猫粮数量。

参考代码

header.h

```
int maxFood(int a[], int len);
```

header.c

```
#include "header.h"

int maxFood(int arr[], int n) {
    int max_sum = 0;
    int current_sum = 0;

    for (int i = 0; i < n; i++) {
        current_sum += arr[i];

        if (current_sum > max_sum) {
            max_sum = current_sum;
        }

        if (current_sum < 0) {
            current_sum = 0;
        }
    }

    return max_sum;
}
```

main.c

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

```c
#include "header.h"
int main()
{
    int a[100];
    int len;
    scanf("%d", &len);
    int i;
    for(i=0; i<len; i++){
        scanf("%d", &a[i]);
    }
    printf("%d", maxFood(a, len));
    return 0;
}
```

代码解释

这其实是一个“最大子数组和”的问题，经典解法为“Kadane’s算法”，它能在线性时间内完成求解。Kadane算法的核心思想是通过迭代数组元素，逐步累计子数组的和，如果累计的和在某一时刻变成负数，就放弃当前子数组，重新开始计算，因为负数只会减少总和。

1. 初始化两个变量：
    - `max_sum`：用于存储找到的最大和，初始值为0。
    - `current_sum`：用于追踪当前的子数组和，初始值为0。
2. 遍历数组：
    - 对于每个元素，将其添加到`current_sum`。
    - 如果`current_sum`比`max_sum`大，更新`max_sum`。
    - 如果`current_sum`变为负数，说明这个子数组的和对后续求最大值没有帮助，将`current_sum`重置为0。（举个例子，如果数组的第一个数就是负数，那么在遍历到第二个索引时便会重置`current_sum`时便会重置）

最终，`max_sum`就是猫咪能获得的最多猫粮数。

另外，本题中定义的代码架构（先定义头文件 *.h* ，再在 *.c* 文件中实现）是 C/C++ 的推荐代码架构，这样组织一个项目，便于维护和拓展。

T2　螺旋遍历二维数组

题目

给定一个m x n大小的整数矩阵（m行，n列），按顺时针螺旋的顺序打印矩阵中的所有元素。

例如一个3 x 3 的 矩阵：

1 2 3

4 5 6

7 8 9

输出应为：

1 2 3 6 9 8 7 4 5

数据规模

`1 <= n,m <= 10`，矩阵中任意元素都满足`|val|<=100`。

提示

只需完成函数 void spiralOrder(int **matrix,int m,int n)，该函数参数matrix是一个二维矩阵，用二维数组进行存储，可以使用matrix[0][0]在该函数中访问矩阵第一行第一列的数据，m表示矩阵的行数，n表示矩阵的列数。该函数需要按照题意顺时针螺旋打印矩阵内容。

参考代码

solution.h

```
void spiralOrder(int **matrix, int m, int n);
```

solution.c

```
#include "solution.h"

void spiralOrder(int **matrix, int m, int n) {
    int top = 0, bottom = m - 1; // 上边界和下边界
    int left = 0, right = n - 1; // 左边界和右边界

    while (top <= bottom && left <= right) {
        // 从左到右遍历上边界
        for (int i = left; i <= right; i++) {
            printf("%d ", matrix[top][i]);
        }
        top++; // 上边界下移

        // 从上到下遍历右边界
        for (int i = top; i <= bottom; i++) {
            printf("%d ", matrix[i][right]);
        }
        right--; // 右边界左移

        // 从右到左遍历下边界（如果还在边界内）
        if (top <= bottom) {
            for (int i = right; i >= left; i--) {
                printf("%d ", matrix[bottom][i]);
            }
            bottom--; // 下边界上移
        }

        // 从下到上遍历左边界（如果还在边界内）
        if (left <= right) {
            for (int i = bottom; i >= top; i--) {
                printf("%d ", matrix[i][left]);
            }
            left++; // 左边界右移
        }
    }
    printf("\n"); // 打印新行以便清晰显示
}
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "solution.h"
int main()
{
    int m,n,i,j;
    scanf("%d%d",&m,&n);
    int **matrix=(int**)malloc(sizeof(int*)*m);
    for(i=0; i<m; i++)
        matrix[i]=(int*)malloc(sizeof(int)*n);
    for(i=0; i<m; i++)
    {
        for(j=0; j<n; j++)
        {
            scanf("%d",&matrix[i][j]);
        }
    }
    spiralOrder(matrix,m,n);
    for(i=0; i<m; i++)
        free(matrix[i]);

    free(matrix);
    return 0;
}
```

代码解释

1. 定义边界：我们定义了四个变量`top`、`bottom`、`left`、`right`来表示矩阵的四个边界，分别初始化为0、`m-1`、0和`n-1`。
2. 螺旋打印循环：
    - 使用`while (top <= bottom && left <= right)`来控制循环，确保打印操作在边界内。
    - 从左到右遍历`top`边界：从`left`到`right`打印`matrix[top][i]`，然后`top++`。
    - 从上到下遍历`right`边界：从`top`到`bottom`打印`matrix[i][right]`，然后`right--`。
    - 从右到左遍历`bottom`边界（如果`top <= bottom`）：从`right`到`left`打印`matrix[bottom][i]`，然后`bottom--`。
    - 从下到上遍历`left`边界（如果`left <= right`）：从`bottom`到`top`打印`matrix[i][left]`，然后`left++`。

这道题只需理解二维数组的有关操作即可轻松完成。

T3　压缩字符串

题目

Description

小G在玩一个游戏，利用字符重复出现的次数，编写一种方法，实现基本的字符串压缩功能。比如，字符串aabcccccaaa会变为a2b1c5a3，假如压缩后的字符串的长度不小于原字符串的长度，则输出原来的字符串。
字符串中只包含小写英文字母（a至z），每个字母的重复次数为1到9次，字符总长度为1~100。

Hint

strlen()可以用来求一个字符串的长度，需要include头文件<string.h>

Input

一行，字符串

Output

输出压缩的字符串或者原字符串

参考代码

main.c

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

void compressString(char *str) {
    int len = strlen(str);
    char compressed[200];
    int index = 0;

    int count = 1;

    for (int i = 0; i < len; i++) {
        if (i + 1 < len && str[i] == str[i + 1]) {
            // 当前字符与下一个字符相同，增加计数
            count++;
        } else {
            // 当前字符与下一个字符不同，添加字符和计数到compressed
            compressed[index++] = str[i];
            compressed[index++] = count + '0';
            count = 1;
        }
    }

    // 添加字符串结束符
    compressed[index] = '\0';

    // 判断压缩字符串是否比原字符串短
    if (strlen(compressed) < len) {
        printf("%s\n", compressed);
    } else {
        printf("%s\n", str);
    }
}

int main() {
```

```
    char str[101];
    scanf("%s", str);
    compressString(str);
    return 0;
}
```

代码解释

1. 输入和初始化：
    - 使用 `scanf` 读取输入字符串 `str`。
    - 使用 `strlen()` 函数获取原字符串长度 `len`。
    - 定义 `compressed` 数组来存储压缩后的字符串，`index` 用于指向 `compressed` 中的位置。
    - `count` 用于记录当前字符的连续出现次数。
2. 遍历字符串：
    - 循环从 `0` 到 `len - 1`，检查每个字符。
    - 如果当前字符与下一个字符相同，则增加 `count`。
    - 如果当前字符与下一个字符不同，则：
        - 将当前字符和计数值追加到 `compressed` 中。使用 `compressed[index++] = str[i]` 将字符加入，使用 `compressed[index++] = count + '0'` 将计数值转为字符加入（`count + '0'` 将整数转换为字符，如 `2` 变成字符 `'2'`）。
        - 重置 `count` 为1，准备统计下一个字符的重复次数。
3. 结束处理：
    - 在 `compressed` 字符串末尾添加空字符 `\0`，标志字符串结束。
    - 比较 `compressed` 和原字符串的长度，如果 `compressed` 长度更小，则输出压缩结果，否则输出原字符串。

T4　大数运算

题目

输入两个非负整数a和b，输出两个非负整数的和(a+b)。

数据规模

0<=a,b<= 1099−1

参考代码

main.c

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

void bigNumberAddition(char *a, char *b) {
    int len_a = strlen(a);
    int len_b = strlen(b);
    int max_len = len_a > len_b ? len_a : len_b;
```

```
    char result[max_len + 2];  // +2是为了处理可能的进位以及末尾的\0
    int carry = 0;  // 进位
    int index = 0;  // 结果数组的索引

    // 从低位到高位逐位相加
    for (int i = 0; i < max_len; i++) {
        int digit_a = (i < len_a) ? a[len_a - 1 - i] - '0' : 0; // 取a的当前位
        int digit_b = (i < len_b) ? b[len_b - 1 - i] - '0' : 0; // 取b的当前位
        int sum = digit_a + digit_b + carry;  // 当前位的和

        result[index++] = (sum % 10) + '0';  // 保存当前位的结果
        carry = sum / 10;  // 更新进位
    }

    // 如果有剩余进位
    if (carry) {
        result[index++] = carry + '0';
    }

    result[index] = '\0';  // 添加字符串结束符

    // 反转结果字符串
    for (int i = 0; i < index / 2; i++) {
        char temp = result[i];
        result[i] = result[index - 1 - i];
        result[index - 1 - i] = temp;
    }

    printf("%s\n", result);  // 输出最终结果
}

int main() {
    char a[102], b[102];  // 最大长度为100，加上末尾的\0
    scanf("%s %s", a, b);
    bigNumberAddition(a, b);
    return 0;
}
```

代码解释

1. 输入和初始化：
   - 读取两个大数字符串 `a` 和 `b`。
   - 计算两个字符串的长度，并找出较长的长度作为 `max_len`。
   - `result` 数组用于存储加法结果，长度为 `max_len + 2`，以防进位导致长度增加。
   - `carry` 用于保存进位。
2. 逐位相加：
   - 从两个字符串的末尾开始逐位相加（如果一个字符串已遍历完，则对应位置的值为0）。
   - 计算每一位的和，将个位加入 `result`，更新进位。
3. 处理进位和结束符：
   - 若最后仍有进位，则将进位加到 `result`。

- 将 `result[index]` 置为 `\0`，表示字符串结束。

4. 反转结果：
   - 因为 `result` 从低位到高位填充，因此需要反转使其符合从高位到低位的顺序。
5. 输出结果：最终输出 `result`。