Experiment12-董皓彧

环境：

```
gcc.exe (x86_64-win32-seh-rev1, Built by MinGW-Builds project) 13.2.0
Visual Stdio Code 1.84.2
```

作业仓库地址：
https://github.com/FHYQ-Dong/Tsinghua-Program-Design-Assignments/tree/main/Experiment12

必做题

Experiment12-1

题目：

> 定义一个指向字符串的指针数组，用一个函数完成 N 个不等长字符串的输入，使得指针数组元素依次指向每一个输入的字符串。设计一个完成 N 个字符串按升序的排序函数（在排序过程中，要求只交换指向字符串的指针，不交换字符串）。在主函数中实现对排序后的字符串的输出。假设已知字符串的最大为 80 字节，根据实际输入的字符串长度来分配存储空间。

输入格式：

```
共 N+1 行：
    第 1 行，一个整数 N，表示有 N 个字符串；
    第 2~N+1 行，每行一个字符串，字符串长度不超过 80 个字符
```

输出格式：

```
共 N 行，按升序排列的字符串
```

代码：

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef char bool;
#define true 1
#define false 0
typedef char* string;

void readstr(string* c, int N) {
    for (int i = 0; i < N; i++) {
        c[i] = (string)malloc(100 * sizeof(char));
        memset(c[i], 0, 100 * sizeof(char));
        scanf("%s", c[i]);
    }
    return;
}
```

```
bool cmp(string a, string b) {
    int i = 0;
    while (a[i] != '\0' && b[i] != '\0') {
        if (a[i] < b[i]) return true;
        if (a[i] > b[i]) return false;
        i++;
    }
    if (a[i] == '\0') return true;
    return false;
}

void swap(string* a, string* b) {
    string temp = *a;
    *a = *b;
    *b = temp;
    return;
}

void qsortstr(string* begin, string* end, bool (*cmp)(string a, string b)) {
    if (begin >= end) return;
    string* pivot = begin;
    string* left = begin;
    string* right = end - 1;
    while (left < right) {
        while (left < right && cmp(*pivot, *right))
            right--;
        while (left < right && cmp(*left, *pivot))
            left++;
        if (left < right) swap(left, right);
    }
    swap(pivot, left);
    qsortstr(begin, left, cmp);
    qsortstr(left + 1, end, cmp);
    return;
}

int main() {
    int N; scanf("%d", &N);
    string* c = (string *)malloc(N * sizeof(string *));
    readstr(c, N);

    qsortstr(c, c + N, cmp);
    for (int i = 0; i < N; i++) printf("%s\n", c[i]);

    for (int i = 0; i < N; i++) free(c[i]);
    free(c);
    return 0;
}
```

输入1：

```
5
1234567890
qwertyuiop
asdfghjkl
zxcvbnm
9q3eofhwguiveulr
```

输出1:

```
1234567890
9q3eofhwguiveulr
asdfghjkl
qwertyuiop
zxcvbnm
```

输入2:

```
3
1234567890
qwertyuiop
asdfghjkl
```

输出2:

```
1234567890
asdfghjkl
qwertyuiop
```

输入3:

```
10
ySAjvCSD7So@m*^LM5iSk7!#6%&S5i*3
a6@79&r@Q#%pYU6VL%WG43dB4769@vUn
@o4y7j8K%&@F39Pejze2m^
7WgwPhgt3wR8C2fG7B3s!#7n2uv&$rd6Ncs4*&hY*NJ*nYtkWAym%A6d54HhrT8!
HY47428BTT$CzpPyd8BW45%5!#!mZ%U#5P*E&5z!&5T%eFczd84g&tP4EGY^SbztmLJ&o^i5hcrT@
7eJ5#38$5
28934ywrheuiofb
vnb39g584werufh9383hiAiajgkfh
*(&$ITYEFOWHLUKdghe9p8rtyugj38aeioafhb
103928ry9t4h3guriosdd
```

输出3:

```
*(&$ITYEFOWHLUKdghe9p8rtyugj38aeioafhb
103928ry9t4h3guriosdd
28934ywrheuiofb
7WgwPhgt3wR8C2fG7B3s!#7n2uv&$rd6Ncs4*&hY*NJ*nYtkWAym%A6d54HhrT8!
7eJ5#38$5
@o4y7j8K%&@F39Pejze2m^
HY47428BTT$CzpPyd8BW45%5!#!mZ%U#5P*E&5z!&5T%eFczd84g&tP4EGY^SbztmLJ&o^i5hcrT@
a6@79&r@Q#%pYU6VL%WG43dB4769@vUn
vnb39g584werufh9383hiAiajgkfh
ySAjvCSD7So@m*^LM5iSk7!#6%&S5i*3
```

输入4：

```
2
qwertyuiop
1234567890
```

输出4：

```
1234567890
qwertyuiop
```

输入5：

```
1
1234567890
```

输出5：

```
1234567890
```

Experiment12-2

题目：

```
编写程序，求一个 N×N 方阵的第 i 对角线的元素之和。给定方阵 a[10][10] = {0, 1, 2, ..., 99}
```

输入格式：

```
无
```

输出格式：

```
共 1 行，10 个数，分别为第 i 对角线的元素之和
```

代码：

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

```
int column_index(int i, int n, int N) { return i+n <= N ? i+n : (i+n)%N; }

int get_sum(int a[11][11], int n) {
    int sum = 0;
    for (int i=1; i<=10; ++i) sum += *(*(a+i) + column_index(i, n, 10));
    return sum;
}

int main() {
    int a[11][11] = {0};
    for (int i=1; i<=10; ++i) for (int j=1; j<=10; ++j) *(*(a+i)+j) = (i-1)*10 +
(j-1);
    for (int n=0; n<10; ++n) printf("%d\n", get_sum(a, n));
    return 0;
}
```

输入1：

```

```

输出1：

```
495
495
495
495
495
495
495
495
495
495
```

Experiment12-3

题目：

```
通常一个基于转义符的 emoji 表情输入由以下三部分构成：转义符 + 表情名称 + 终止符以新浪微博为例，当微博正文读取到一个转义符“[”时，它与终止止符“]”之间的文字将作为表情名称在表情库中进行搜索，如果存在匹配表情，则输出显示。
注意，如果在一段语句中存在多个转义符和一个终止符，那么以离终止符最近的一个转义符作为表情的起始标志。
```

输入格式：

```
共 3 行：
    第 1 行：起始标志字符
    第 2 行：终止标志字符
    第 3 行：待处理的字符串
```

输出格式：

共若干行，每行一个表情名称文字，按照输入顺序输出

代码：

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

int main() {
    char beg, end;
    char s[150];
    memset(s, 0, sizeof(s));
    scanf("%c\n%c\n", &beg, &end);
    fgets(s, 150, stdin); // get a line

    char* his = NULL;
    for (char* p = s; *p != '\0'; ++p) {
        if (*p == beg) his = p;
        if (*p == end) {
            if (his == NULL) {
                printf("Error: no matching '%c' for '%c' at %d\n", beg, end, p-
s);
                return 0;
            }
            for (char* q = his+1; q < p; ++q) printf("%c", *q);
            printf("\n");
            his = NULL;
        }
    }
    return 0;
}
```

输入1：

```
*
#
Time for lunch. *greedy# Hope a big meal.
```

输出1：

```
greedy
```

输入2：

```
*
#
*happy*smile#
```

输出2：

```
smile
```

输入3：

```
[
]
Emmm... You use [Grin] instead of [Smile] when you are really happy in wechat.
```

输出3：

```
Grin
Smile
```

输入4：

```
[
]
[a[b[c[d[e[f[g][][[[][f]
```

输出4：

```
g


f
```

输入5：

```
a
b
a[b[c[d[e[f[g[][[[][f]aa
```

输出5：

```
[
```

选做题

Optional-Experiment12-1

题目：

```
在图像的基于 DCT 变换的压缩中，通常对 DCT 变换后的系数矩阵进行 Zig-Zag 扫描。所谓 Zig-Zag 扫描，又名“之”字型扫描，即从矩阵的第一行第一列系数开始，按照“之”字形方向进行系数读取。以示例 3 阶矩阵 {1，2，3，4，5，6，7，8，9} 为例，经过 Zig-Zag 扫描后，输出数据顺序为 1 2 4 7 5 3 6 8 9
```

输入格式：

```
共 N+1 行：
    第 1 行，一个整数 N，表示矩阵的阶数；
    第 2~N+1 行，每行 N 个整数，表示矩阵的系数
```

输出格式：

```
一行，Zig-Zag 扫描后的系数，以空格分隔
```

代码：

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

int main() {
    int n; scanf("%d", &n);
    int** a = (int**)malloc(sizeof(int*) * (n+1));
    for (int i=1; i<=n; ++i) {
        *(a+i) = (int*)malloc(sizeof(int) * (n+1));
        for (int j=1; j<=n; ++j) scanf("%d", *(a+i)+j);
    }

    /* Zig-Zag
        * 二维数组 a 从 a[1][1] 开始使用
        * 以下枚举 i 的意义是枚举这样一条左上-右下对角线，在这条线上的元素 a[x][y] 满足
x+y=i
    */
    for (int i=2; i<=n+1; ++i) { // 左上三角：j 为每个元素的横坐标
        if (i%2 == 1) for (int j=1; j<=i-1; ++j) printf("%d ", a[j][i-j]);
        else for (int j=i-1; j>=1; --j) printf("%d ", a[j][i-j]);
    }
    for (int i=n+2; i<=2*n; ++i) { // 右下三角：j 为每个元素的纵坐标
        if (i%2 == 1) for (int j=n; i-j<=n; --j) printf("%d ", a[i-j][j]);
        else for (int j=i-n; j<=n; ++j) printf("%d ", a[i-j][j]);
    }
    return 0;
}
```

输入1：

```
3
1 2 3
4 5 6
7 8 9
```

输出1：

```
1 2 4 7 5 3 6 8 9
```

输入2：

```
4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

输出2：

```
1 2 5 9 6 3 4 7 10 13 14 11 8 12 15 16
```

输入3：

```
2
1 2
3 4
```

输出3：

```
1 2 3 4
```

输入4：

```
5
25 24 23 22 21
20 19 18 17 16
15 14 13 12 11
10 9 8 7 6
5 4 3 2 1
```

输出4：

```
25 24 20 15 19 23 22 18 14 10 5 9 13 17 21 16 12 8 4 3 7 11 6 2 1
```

输入5：

```
1
1
```

输出5：

```
1
```

Optional-Experiment12-2

题目：

```
假设将下列程序生成可执行文件 test.exe ，使用命令行运行 test FINAL EXAM ，则程序的输出结果是
```

输入格式：

```
无
```

输出格式：

```
输出结果，具体解析见代码注释
```

代码：

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    char **p = NULL;
    for (p=argv; argc--; ++p)
        printf("%c%s", **p, *p);
        /*
            * *argv[] = {"test.exe", "FINAL", "EXAM"};
            * 1. p = argv[0] = "test.exe"
            *   **p = p[0][0] = 't', *p = p[0] = "test.exe";
            * 2. p = argv[1] = "FINAL":
            *   **p = p[1][0] = 'F', *p = p[1] = "FINAL";
            * 3. p = argv[2] = "EXAM":
            *   **p = p[2][0] = 'E', *p = p[2] = "EXAM";
        */
       // result = ttest.exeFFINALEEXAM
    return 0;
}
```

输入1：

```

```

输出1：

```
ttest.exeFFINALEEXAM
```