信手相连——面向过程的程序设计 (C 语言)

4. 文件与高级程序设计技术（递归）

崔冠宇

2018202147

https://github.com/GuanyuCui/RUC-infohands-2020

信息学院
中国人民大学

2020-2021 学年秋季学期
(12.6, 18:00-19:30, 教二 2114)

目录

课程目标与进度

基础知识回顾

提高知识

进度

我们的课程已经接近尾声。本章的主题是文件与高级程序设计技术（主要是递归）。

- ✓ 基础知识（计算机结构、代码风格与规范……）；
- ✓ 数据类型（长短整、单双精度浮点、字符……）；
- ✓ 控制结构（顺序、选择、循环）；
- ✓ 数组；
- ✓ 函数；
- ✓ 指针；
- ✓ 自定义类型（枚举、联合体、结构体）；
- → 文件；
- → 提高知识（递归、动态规划、数据结构、大作业指导）。

动态内存分配

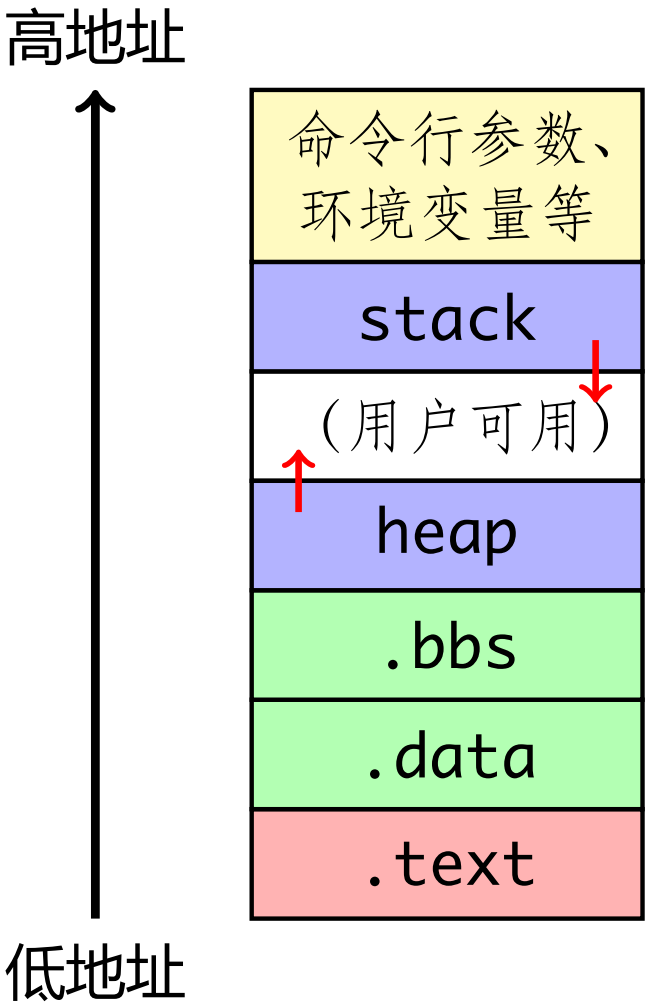
之前大家定义的数组在运行时必须是定长的，就算是"变长数组"(VLA) 也只是允许变量作为数组长度而已。但是如果遇到不能提前判断元素个数，而运行时又发现数组空间不够该怎么办呢？于是就引出了动态内存分配的方法。

动态内存管理相关函数定义于 `<stdlib.h>`，主要有：

- `malloc`：分配内存；
- `calloc`：分配并清零内存；
- `realloc`：扩充之前分配的内存块；
- `free`：归还之前分配的内存。

一般而言，都是用指针的方式操作动态分配得到的内存，因此程序员要格外小心。由于动态分配的内存不会由程序自动释放，所以动态分配得到的内存需要程序员手动释放。

回顾——变量（在程序中）的内存布局



1. 栈区（stack）：自动分配释放，存储函数参数，局部变量等，操作方法类似数据结构中的栈；
2. 堆区（heap）：一般由程序员申请和释放，与数据结构中的堆没有关系，分配方式类似链表；
3. 未初始化静态 / 全局变量区（.bbs）：在程序编译时分配；
4. 已初始化静态 / 全局变量区（.data）：在程序编译时分配；
5. 程序代码区（.text）：存储函数的二进制代码。

文件

文件，简单而言就是存放在文件系统中的一堆数据。*nix 系统的设计哲学是——

> 万物皆文件。[1]

因此，在这些系统上，硬件设备也是文件（如标准输入 stdin、标准输出 stdout）。除去大家熟悉的磁盘上的普通文件之外，包括一些设备在内的文件被称为特殊文件。

在 C 语言中，文件操作是以文件结构体 FILE 和文件结构体指针 FILE * 实现的；在 C++ 中，输入输出以及文件操作都是流式的，输入输出分别有 istream 和 ostream 类，文件有 fstream 类型。

[1] 顺便一提，**ZFC** 的哲学是万物皆集合。

C 语言文件操作函数

文件访问 <stdio.h>：

- fopen：打开文件；
- freopen：以不同名称打开既存的文件流；
- fclose：关闭文件；

无格式输入/输出 <stdio.h>：

- fgetc/getc：从文件流获取一个字符；
- fgets：从文件流获取一个字符串；
- fputc/putc：将一个字符写入文件流；
- fputs：将一个字符串写入文件流；
- getchar/gets/putchar/puts：上述四项的 stdin/stdout 版本。
- ungetc：将一个字符送回文件流。

C 语言文件操作函数

有格式输入/输出 <stdio.h>：

- scanf/fscanf/sscanf：从文件流获取一个字符；
- printf/fprintf/sprintf/snprintf：从文件流获取一个字符串；
- fputc/putc：将一个字符写入文件流；
- fputs：将一个字符串写入文件流；
- getchar/gets/putchar/puts：上述四项的 stdin/stdout 版本。
- ungetc：将一个字符送回文件流。

C 语言文件操作函数

文件位置 <stdio.h>：

- ftell：返回当前的文件位置指示值；
- fseek：将文件位置指示符移动到文件中的指定位置；
- rewind：将文件位置指示器移动到文件首；

错误处理 <stdio.h>：

- feof：检查文件结尾；

递归问题——概述

递归（recursion），是指函数定义时使用自身的方法。形象地说就是“套娃”。

在数理逻辑领域，递归论是数理逻辑“四论”之一（其他三论是集合论、模型论和证明论），具有重要的意义。许多数学概念如部分函数的定义，以及计算机科学中可计算性等概念都需要递归（如加法函数、乘法函数等）。

例

设 $S(x)$ 是（Peano 公理规定的）后继函数，自然数 $\mathbb{N}$ 上的加法函数 $f(x,y)=x+y$ 定义如下：

$$f(0,y)=y(y\in\mathbb{N})$$
$$f(S(x),y)=S(f(x,y))(x,y\in\mathbb{N})$$

根据自然数上的递归定理，f 被唯一决定。

递归问题——递归与数列（阶乘函数）

例子：阶乘函数

阶乘函数 $f(n) = n!$ 定义如下：

$$f(0) = 1$$
$$f(n) = nf(n-1)(n \geq 1)$$

请编写程序计算 $f(n)$。

```
1 unsigned long long factorial(unsigned n)
2 {
3     if(n == 0)
4         return 1;
5     else return n * factorial(n - 1);
6 }
```

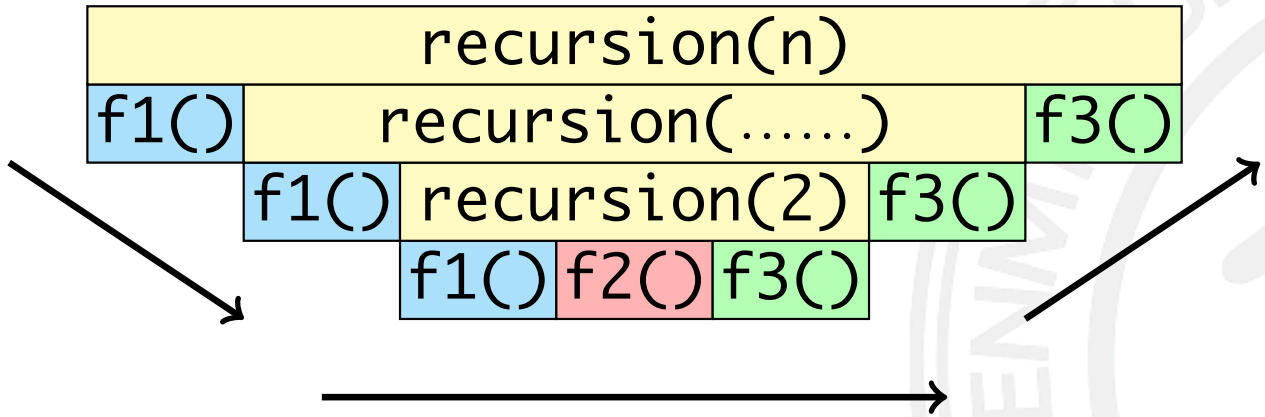
递归问题——一般形式

递归函数的一般写法是：

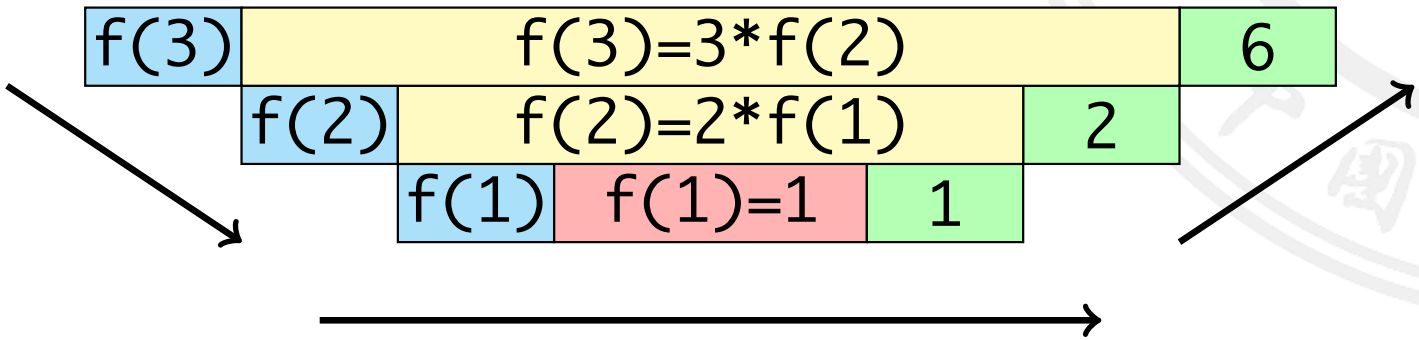
```
1  func recursion(params)
2  {
3      f1(); // 递归前操作，可省略
4      if(到达边界)
5      {
6          f2(); // 边界操作
7      }
8      else
9      {
10         recursion(params); // 递归调用
11     }
12     f3(); // 递归后操作，可省略
13 }
```

递归问题——执行过程

上述递归函数的执行顺序是：



阶乘函数 f(3) 的运行过程可以类似表示：



递归问题——递归与数列（斐波那契数列）

例子：斐波那契数列

斐波那契数列 F_n 定义如下：

$$F_0 = 0, F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2} (n \geq 2)$$

请编写程序计算 F_n。

编写递归函数的要领是，先写出不依赖于函数本身的边界条件，然后再写出递归表达式。

递归问题——递归与数列（阿克曼函数）

例子：阿克曼函数

阿克曼函数 $A(m,n)$ 定义如下：

$$A(m,n)=\begin{cases} n+1, & \text{若} m=0 \\ A(m-1,1), & \text{若} m>0 \text{且} n=0 \\ A(m-1,A(m,n-1)), & \text{若} m>0 \text{且} n>0 \end{cases}$$

请编写程序计算 $A(m,n)$。

阿克曼函数增长速度极快，$A(1,n)=n+2$，$A(2,n)=2n+3$，$A(3,n)=2^{n+3}-3$，$A(4,2)=2^{65536}-3$。它是一个非原始递归的函数，打破了之前人们普遍的想法——"递归函数都是原始递归的"。某些数据结构（如并查集）的复杂度分析中，用到了它的反函数 $\alpha(n)$。

递归问题——递归与数列（上楼梯）

例子：上楼梯

你面前有 N 阶楼梯，你可以选择一步上一阶，也可以选择一步上两阶。

请编写程序计算 N 阶楼梯的上楼方法数。

分析：上一层台阶后，变为还剩 $N-1$ 阶的情况；上两层台阶后，变为还剩 $N-2$ 阶台阶的情况。

边界条件：仅有一层台阶，仅有两层台阶。

递归问题——递归与分治（二分搜索）

例子：有序数组上的搜索

给定升序的数组，要求快速（$O(\log n)$）搜索给定元素 x。

```
1  int binSearch(int A[], int l, int h, int x)
2  {
3      if(l == h)
4          if(A[l] == x) return l;
5          else return -1;
6      int m = (l + h) / 2;
7      if(A[m] == x) return m;
8      else if(A[m] < x)
9          return binSearch(A, m + 1, h, x);
10     else return binSearch(A, l, m - 1, x);
11 }
```

递归问题——典型递归问题（质因数分解）

例子：质因数分解

给定一个大于 1 的自然数，将其分解为若干质因数的乘积（要求质因数从小到大）。

提示：先分解出一个最小的因数，然后就转化为子问题。（思考一下为什么？）

边界条件：质数。

递归问题——典型递归问题（质因数分解）

分解最小因数：

```
1  #include <stdio.h>
2  #include <math.h>
3  typedef unsigned long long ull;
4  ull minFactor(ull n)
5  {
6      for(ull i = 2; i <= sqrt(n); i++)
7          if(n % i == 0)
8              return i;
9      return n;
10 }
```

递归问题——典型递归问题（质因数分解）

递归分解质因数：

```
11 void decompose(ull n)
12 {
13     ull factor = minFactor(n);
14     if(factor == n)
15         printf("%llu", n);
16     else
17     {
18         printf("%llu*", factor);
19         decompose(n / factor);
20     }
21 }
```

递归问题——典型递归问题（质因数分解）

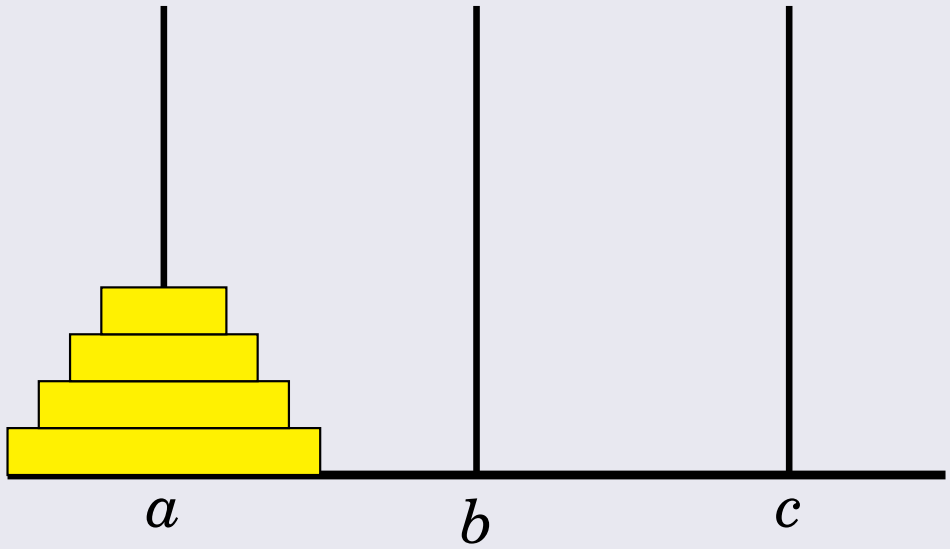
主函数：

```
22 int main(int argc, char *argv[])
23 {
24     ull a, b;
25     scanf("%llu%llu", &a, &b);
26     for(ull i = a; i <= b; i++)
27     {
28         printf("%llu=", i);
29         decompose(i);
30         printf("\n");
31     }
32     return 0;
33 }
```

递归问题——典型递归问题（汉诺塔）

例子：汉诺塔

如下图，现在有三根柱子 a、b 以及 c，a 柱上叠放着 n 个圆盘，它们的大小依次增大。请你按照汉诺塔的规则（一次只能挪动一个圆盘，大圆盘不能盖在小圆盘上），输出圆盘的移动步骤（格式：Move disk x from x to x）。



递归问题——典型递归问题（汉诺塔）

```
1  #include <stdio.h>
2  void hanoi(int n, char from, char h, char to)
3  {
4      if(n == 1)
5      {
6          printf("Move disk 1 from %c to %c\n",
       from, to);
7          return;
8      }
9      hanoi(n - 1, from, to, h);
10     printf("Move disk %d from %c to %c\n", n,
       from, to);
11     hanoi(n - 1, h, from, to);
12 }
```

递归问题——典型递归问题（汉诺塔）

```
13 int main(int argc, char *argv[])
14 {
15     int n;
16     scanf("%d", &n);
17     hanoi(n, 'a', 'b', 'c');
18     return 0;
19 }
```

递归问题——典型递归问题（图搜索）

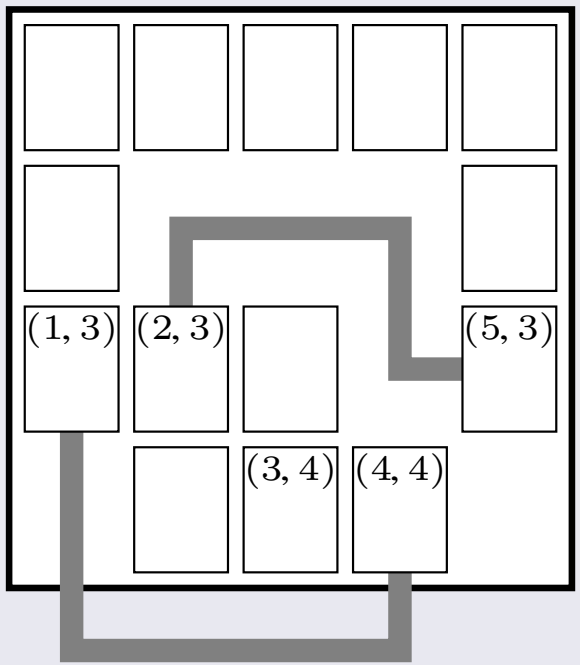
例子：连连看游戏 (2018 期末，6)

连连看游戏在一个分割成 $w \times h$ 个正方格子的矩形板上进行。如图所示，每个正方格子上可以有一张游戏卡片，当然也可以没有。当下面的情况满足时，我们认为两个游戏卡片之间有一条路径相连：

1. 路径中只包含水平或者竖直的直线段；
2. 路径不能穿过别的游戏卡片；
3. 允许路径临时的离开矩形板。

递归问题——典型递归问题（图搜索）

例如，下图的例子中 (1, 3) 和 (4, 4) 处的游戏卡片是可以相连的。而在 (2, 3) 和 (3, 4) 处的游戏卡是不相连的，因为连接他们的每条路径都必须要穿过别的游戏卡片。



递归问题——典型递归问题（图搜索）

请你编程判断是否存在一条满足题意的路径能连接给定的两个游戏卡片，且路径中包含的线段数不大于 10。如果可以找到这样的路径，找到连接这两个卡片的所有路径中包括线段数最少的路径，输出最少的线段数；如果不能相连，输出“impossible”(注意如果连接 2 张游戏卡片的所有路径中包含的线段均大于 10 也应输出 impossible)。

递归问题——典型递归问题（图搜索）

输入格式

第一行包括两个整数 w 和 h $(1 \le w, h \le 75)$，分别表示矩形板的宽和长。

下面的 h 行，每行包括 w 个字符，表示矩形板上的游戏卡片分布情况。使用'X'表示这个地方有一游戏卡片；使用空格表示这个地方没有游戏卡片。

之后一行一个整数 n，表示后面要输入 n 对卡片的位置。

接下来的 n 行上，每行包括 4 个整数 x_1, y_1, x_2, y_2 $(1 \le x_1, x_2 \le w; 1 \le y_1, y_2 \le h)$，给出两张卡片在矩形板上的位置 (x_1, y_1) 和 (x_2, y_2)。

注意：矩形板左上角的坐标是 (1, 1)。输入保证这两个游戏卡片所处的位置是不相同的。

递归问题——典型递归问题（图搜索）

输出格式

n 行。

每行输出一对卡片的连接情况。如果找到连接这两个卡片的所有路径中包括线段数最少的路径，输出最少的线段数；如果不能相连或者连接卡片的所有路径中包含的线段均大于 10，输出“impossible”。

递归问题——典型递归问题（图搜索）

评测样例

输入：

```
5 4
XXXXX
X   X
XXX X
 XXX
3
2 3 5 3
1 3 4 4
2 3 3 4
```

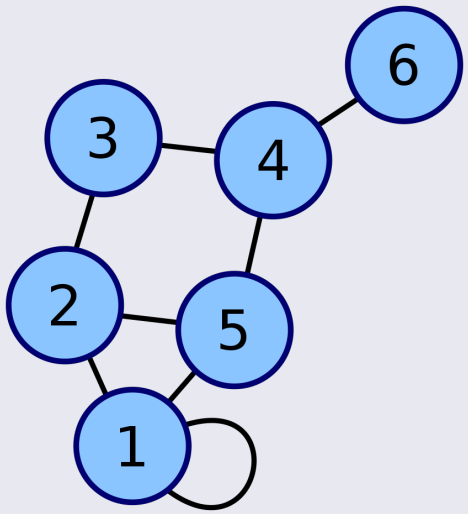
输出：

```
4
3
impossible
```

递归问题——典型递归问题（图搜索）

图的概念

图 G 是一个二元组 (V,E)。其中 V 是顶点集合，E 是边集合。E 中的元素的形式为 (u,v)，$u,v \in V$。



递归问题——典型递归问题（图搜索）

遍历图节点主要有以下两种方法——深度优先搜索（Depth First Search，DFS）和广度优先搜索（Breadth First Search，BFS）。下面分别介绍这两种搜索方式，先介绍 DFS：

深度优先搜索（DFS）

简单而言就是“一棵树吊死”。任意选定图中一顶点，尽可能深的搜索分支。当节点 v 的邻居都已被探寻过，搜索将回溯到发现节点 v 的那条边的起始节点。这一过程一直进行到已发现从源节点可达的所有节点为止。

根据 DFS 的含义，当需要尽快找到一条可行道路时，应当选择 DFS。

递归问题——典型递归问题（图搜索）

由于 DFS 的特性（分别对每一个没有探索过的邻居进行 DFS），所以可以用递归实现。

深度优先搜索（递归实现）：

```
1 DFS-RECURSIVE(G, v):
2     visited[v] = true
3     for v 的每一个邻居 u:
4         if not visited[u]:
5             对 u 进行操作
6             visited[u] = true
7             DFS(G, u)
```

但是递归实现版本有一个问题，递归是需要运行栈和保护现场的开销的，对于大规模图的 DFS 可能有栈溢出的风险。但我们可以手动改为栈实现的方式，减少递归调用的开销。

递归问题——典型递归问题（图搜索）

由于栈的后入先出的特性，所以 DFS 可以用栈实现。

深度优先搜索（栈实现）：

```
1  DFS-STACK(G, v):
2      Stack s
3      s.push(v)
4      visited[v] = true
5      while not s.empty():
6          node = s.top()
7          s.pop()
8          对 node 进行操作
9          for node 的每一个邻居 u:
10             if not visited[u]:
11                 visited[u] = true
12                 s.push(u)
```

递归问题——典型递归问题（图搜索）

广度/宽度优先搜索（BFS）

简单而言是"广撒网"。对于一个结点，展开其所有未探索过的邻居，按照展开顺序探索节点，同时再次展开它们未探索过的邻居。

BFS 的特点是，每时每刻探索到的众多路径的长度最多相差一。根据这一特点，当需要尽快找到一条（无权图的）最短路时，应当选择 BFS。

一般而言，BFS 无法使用递归实现，要专门用队列实现。

递归问题——典型递归问题（图搜索）

广度优先搜索（队列实现）：

```
1  BFS-QUEUE(G, v):
2      Queue q
3      q.enqueue(v)
4      visited[v] = true
5      while not q.empty():
6          node = q.front()
7          q.dequeue()
8          对 node 进行操作
9          for node 的每一个邻居 u:
10             if not visited[u]:
11                 visited[u] = true
12                 q.enqueue(u)
```

递归问题——典型递归问题（图搜索）

代码讲解：

- 连连看游戏（DFS，递归，部分超时）——`dfs.c`
- 连连看游戏（BFS，AC）——`bfs.c`
- 哥德巴赫猜想（筛法，AC）——`goldbach.c`

致谢

感谢大家到场支持，也欢迎大家积极提出意见和建议。大家也可以谈谈想要在信手相连的课程中听什么内容，我将酌情做出调整。