# 题目

当且仅当每个相邻位数上的数字 x 和 y 满足 x <= y 时，我们称这个整数是单调递增的。

给定一个整数 n ，返回 小于或等于 n 的最大数字，且数字呈 单调递增 。

示例 1:

输入: n = 10

输出: 9

示例 2:

输入: n = 1234

输出: 1234

示例 3:

输入: n = 332

输出: 299

提示:

0 <= n <= 109

# 分析

要解决“找到小于或等于n的最大单调递增整数”的问题，核心思路是从后向前检查数字的单调性，一旦发现递减位置，就通过“减1+后续补9”的方式调整，确保最终数字满足单调递增且尽可能大。

解题思路：

1、数字转数组：将整数n转换为字符数组（或数字数组），方便逐位修改和检查（字符数组可直接操作每一位数字的ASCII值，无需频繁转换）。

2、从后向前检查单调性：

- 从倒数第二位开始向前遍历，若当前位数字digit[i]大于下一位数字digit[i+1]，说明此处出现递减，需调整。

- 调整规则：当前位数字减1（digit[i]--），并标记“后续所有位需补9”（避免后续位仍存在递减，且补9能保证数字最大）。

3、补9操作：遍历到标记位置后，将后续所有位数字设为9（例如332调整为329后，继续检查发现3>2，再调整为299）。

4、数组转整数：将调整后的字符数组转回整数，即为结果。

代码实现（含详细注释）：

class Solution {

public:

int monotoneIncreasingDigits(int n) {

// 1. 将整数n转换为字符数组，方便逐位操作（处理数字时无需频繁类型转换）

string s = to\_string(n);

int len = s.size();

// flag记录从哪个位置开始，后续所有位需要补9（初始为len，即无需补9）

int flag = len;

// 2. 从后向前检查数字的单调性（从倒数第二位开始，对比当前位与下一位）

for (int i = len - 2; i >= 0; --i) {

// 若当前位 > 下一位，说明出现递减，需要调整

if (s[i] > s[i + 1]) {

s[i]--; // 当前位减1，消除递减

flag = i + 1; // 标记：从下一位开始，后续所有位需补9

}

}

// 3. 将flag及之后的所有位设为9，确保整体单调递增且数字最大

for (int i = flag; i < len; ++i) {

s[i] = '9';

}

// 4. 将调整后的字符数组转回整数，返回结果

return stoi(s);

}

};

代码解释：

1、数字转字符数组：使用to\_string(n)将整数转为字符串，每一位字符直接对应数字的ASCII值（例如'3'的ASCII值为51，减1后为50，即'2'），操作直观且高效。

2、从后向前检查：从倒数第二位开始遍历，是因为调整当前位时，仅需确保与下一位的单调性（下一位及之后的数字已通过之前的遍历保证单调，或已标记为补9），避免重复检查。

3、补9逻辑：标记flag后，后续位补9的原因是：当前位减1后，后续位补9能最大化数字（例如332→329→299，299是小于332的最大单调递增数）。

4、边界处理：对于本身已单调递增的数字（如1234），flag始终为len，无需补9，直接返回原数字；对于n=0等特殊情况，to\_string(0)为"0"，循环不执行，返回0，符合预期。

复杂度分析

- 时间复杂度：O(k)，其中k是整数n的位数（如n=10^9时k=10）。两次遍历均为O(k)，字符数组转整数也为O(k)，整体线性时间。

- 空间复杂度：O(k)，用于存储转换后的字符数组（字符串），空间开销与数字位数成正比。

该方法通过“检查-调整-补9”的逻辑，高效找到符合条件的最大单调递增数，避免了暴力枚举（从n向下遍历直到找到目标数）的低效问题（暴力法在n=10^9时可能遍历次数极多）。