**作业 HW1\* 实验报告**

姓名：张嘉麟 学号：2352595 日期：2024年9月24日

# 实验报告格式要求按照模板（使用Markdown等也请保证报告内包含模板中的要素）

# 对字体大小、缩进、颜色等不做强制要求（但尽量代码部分和文字内容有一定区分，可参考vscode配色）

# 实验报告要求在文字简洁的同时将内容表示清楚

# 报告内不要大段贴代码，尽量控制在20页以内

1. **涉及数据结构和相关背景**

# 题目或实验涉及数据结构的相关背景

**2. 实验内容**

**2.1 轮转数组**

**2.1.1 问题描述**

给定一个整数顺序表nums，将顺序表中的元素向右轮转 k 个位置，其中 k 是非负数。

**2.1.2 基本要求**

输入的第一行为两个整数 n 和 k，其中 n 表示 nums 的元素个数，k 表示需要向右轮转的位置数。

第二行为 n 个整数，表示顺序表 nums 中的元素。

输出为轮转后的顺序表中的元素。

**2.1.3 数据结构设计**

struct BigInt {

string value; // 存储大整数的字符串

BigInt() : value("0") {} // 默认构造函数，初始化为0

// 加法操作

BigInt add(const BigInt &other) {

// 这里实现大整数的加法

}

// 乘法操作

BigInt multiply(int num) {

// 这里实现大整数与整数的乘法

}

};

**2.1.4 功能说明（函数、类）**

\* @brief 计算级数 S = A + 2A^2 + ... + N \* A^N

\* @param N 级数的上限

\* @param A 级数的底数

\* @return 计算得到的结果（大整数）

\*/

BigInt computeSeries(int N, int A) {

BigInt result; // 初始化结果为0

BigInt powerOfA; // 用于存储 A 的幂，初始为 A^0 = 1

powerOfA.value = "1"; // A^0

for (int i = 1; i <= N; i++) {

powerOfA = powerOfA.multiply(A); // 计算 A^i

BigInt term = powerOfA.multiply(i); // 计算 i \* A^i

result = result.add(term); // 累加到结果中

}

return result; // 返回最终结果

}

**2.1.5 其他四种办法**

**方法一：反转数组（原地算法）**

\* @brief 反转数组中的一部分

\* @param nums 要反转的整数向量

\* @param start 开始索引

\* @param end 结束索引

void reverse(std::vector<int>& nums, int start, int end) {

while start < end do

swap nums[start] with nums[end]

increment start by 1

decrement end by 1

end while

}

\* @brief 将向量中的元素向右轮转 k 个位置

\* @param nums 要轮转的整数向量

\* @param k 轮转的步数

void rotate(std::vector<int>& nums, int k) {

let n = length of nums

k = k mod n // 如果 k 大于 n，则只需轮转 k % n 次

// 反转整个向量

call reverse(nums, 0, n - 1)

// 反转前 k 个元素

call reverse(nums, 0, k - 1)

// 反转剩下的元素

call reverse(nums, k, n - 1)

}

**注释说明**

reverse: 用于反转数组的一部分，输入参数为数组和起始及结束索引。

rotate: 实现轮转数组的功能，输入参数为数组和轮转次数 k。

输出结果为原地修改后的数组。

**方法二：模拟轮转（仅改变输出顺序）**

\* @brief 模拟轮转并打印数组

\* @param nums 数组

\* @param k 轮转次数

void printRotatedArray(std::vector<int>& nums, int k) {

let n = length of nums

// 先打印后 k 个元素

for i from k to n - 1 do

print nums[i] followed by a space

end for

// 再打印前 k 个元素

for i from 0 to k - 1 do

print nums[i] followed by a space

end for

// 打印换行符

print newline

}

**注释说明**

printRotatedArray: 按照轮转后的顺序打印数组，输入参数为数组和轮转次数 k。

输出结果为按轮转顺序打印的数组。

**方法三：使用额外空间**

\* @brief 使用额外空间进行轮转

\* @param nums 数组

\* @param k 轮转次数

void rotateUsingExtraSpace(std::vector<int>& nums, int k) {

let n = length of nums

// 创建与原数组相同大小的新数组

create new array aux with size n

// 根据轮转规则填充新数组

for i from 0 to n - 1 do

aux[(i + k) mod n] = nums[i]

end for

// 将新数组内容复制回原数组

for i from 0 to n - 1 do

nums[i] = aux[i]

end for

// 释放新数组内存（如果适用）

deallocate aux

}

**方法四：循环赋值（原地算法）**

\* @brief 使用循环赋值进行原地轮转

\* @param nums 数组

\* @param k 轮转次数

void rotateInPlace(std::vector<int>& nums, int k) {

let n = length of nums

k = k mod n // 如果 k 大于 n，则只需轮转 k % n 次

let count = 0

let start = 0

while count < n do

let current = start

let prev = nums[start]

while true do

let next\_idx = (current + k) mod n

let temp = nums[next\_idx]

nums[next\_idx] = prev

prev = temp

current = next\_idx

increment count by 1

if start equals current then

break

end if

end while

increment start by 1

end while

**注释说明**

notateInPlace: 使用循环赋值进行原地轮转，输入参数为数组和轮转次数 k。

输出结果为原地修改后的数组。

**2.1.6 调试分析（遇到的问题和解决方法）**

**问题**：大整数运算未处理进位情况，导致计算错误。

**解决方法**：在实现加法和乘法函数时，仔细检查每一位的进位，并在计算完成后处理剩余的进位。

**问题**：输入范围超出限制，程序未能正确处理。

**解决方法**：在读取输入时，添加条件检查，确保输入值在有效范围内。

**2.1.7 总结和体会**

通过本次实验，我对大整数的实现和运算有了更深入的理解。在实际实现过程中，如何有效处理字符串中的每一位数字，确保进位正确，是一个重要的挑战。此题的难点在于对高精度运算的需求以及如何高效地实现级数计算。此次实验让我意识到数据类型的选择与算法效率之间的关系，以及在边界条件下的仔细检查。

**2.2 学生信息管理**

**2.2.1 问题描述**

题目要求实现一个顺序表来管理学生信息，其中包括学号和姓名。需要支持的功能有：根据指定学生个数输入学生信息；在指定位置插入新的学生信息；删除指定位置的学生记录；根据姓名或学号查找学生信息；统计表中学生个数。

**2.2.2 基本要求**

输入学生总数n。

输入n行学生信息（学号、姓名）。

支持插入、删除、查找操作，并根据操作结果输出相应的信息。

在操作结束后输出学生总数并退出程序。

**2.2.3 数据结构设计**

使用vector来存储学生信息，这样可以方便地进行插入和删除操作。每个学生信息由一个结构体表示，包含学号和姓名两个字段。

// 学生信息结构体定义

struct Student {

std::string id; // 学号

std::string name; // 姓名

};

**2.2.4 功能说明（函数、类）**

\* @brief 插入学生信息到指定位置

\* @param pos 插入位置

\* @param student 插入的学生信息

\* @return 操作状态码

void insert(int pos, const Student& student);

\* @brief 删除指定位置的学生信息

\* @param pos 删除位置

\* @return 操作状态码

void remove(int pos);

\* @brief 根据姓名查找学生信息

\* @param name 学生姓名

\* @return 学生信息或-1

void checkName(const std::string& name);

\* @brief 根据学号查找学生信息

\* @param id 学号

\* @return 学生信息或-1

void checkNo(const std::string& id);

\* @brief 获取学生总数

\* @return 学生总数

int getSize();

\* @brief 输入学生信息

\* @param n 学生总数

void inputStudents(int n);

**2.2.5 调试分析（遇到的问题和解决方法）**

在调试过程中遇到的主要问题是边界条件的处理，比如插入或删除时位置是否合法。此外，还发现了一些关于输入输出格式的小问题，如多余的空格等。这些问题通过增加条件判断语句和改进输入输出格式得到了解决。

**2.2.6 总结和体会**

通过本次练习，进一步理解了顺序表的工作原理，并学习到了如何使用std::vector来高效地实现顺序表。同时也意识到了在处理用户输入时，必须仔细检查边界条件，防止程序出现意外错误。此外，对于输出格式的严格遵守也是保证程序正确性的关键因素之一。

**2.3 一元多项式的相加和相乘**

**2.3.1 问题描述**

**2.3.2 基本要求**

**2.3.3 数据结构设计**

**2.3.4功能说明（函数、类）**

**2.3.5 调试分析（遇到的问题和解决方法）**

**2.4 求级数**

**2.4.1 问题描述**

**2.4.2 基本要求**

**2.14.3 数据结构设计**

**2.4.4功能说明（函数、类）**

**2.4.5 调试分析（遇到的问题和解决方法）**

**2.5 扑克牌游戏**

**2.5.1 问题描述**

**2.5.2 基本要求**

**2.5.3 数据结构设计**

**2.5.4功能说明（函数、类）**

**2.5.5 调试分析（遇到的问题和解决方法）**

**3. 实验总结**