崇实战队 26 赛季算法组第一次作业报告

一、作业概述

本次作业围绕算法基础编程展开，共完成三个核心编程任务，分别是斐波那契数列相关功能实现、四种排序算法实现以及学生成绩管理系统开发。所有代码均采用 C 语言编写，严格遵循编程规范，实现了各任务要求的功能，并通过测试验证功能可用性。

编程部分

一、输出斐波那契数列

> 了解算法的同学都知道，递归和迭代都能解决复杂的重复问题，二者有着千丝万缕的联系，同时又存在一些区别，具体体现在在实现方式、效率及可读性等方面。递归和迭代在解决问题的方式上有所不同。递归通过函数调用自身实现重复操作，而迭代则通过循环结构实现。在选择使用哪种方法时，需要考虑具体问题的需求和性能要求。例如，如果问题具有自然的递归结构，或者需要简洁的代码表示，则递归可能是更好的选择；反之，如果对性能要求较高，或者需要避免栈溢出的风险，则迭代可能是更优的选择。
>
> 可见，递归和迭代各有优缺点，适用于不同的场景，所以，理解好二者有助于实际编程中做出明智的选择。

1.题目背景

斐波那契数列又称黄金分割数列，是一个经典的数学序列，在自然界和计算机科学中都有广泛应用。该数列由意大利数学家莱昂纳多·斐波那契提出，用于描述兔子繁殖的数学规律。 数列定义：

- F(0) = 0
- F(1) = 1
- F(n) = F(n-1) + F(n-2) (n ≥ 2) 数列示例：0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

题目要求

编写一个完整的程序，实现以下功能：

1. 计算斐波那契数列的第n项
2. 输出斐波那契数列的前n项
3. 计算斐波那契数列的前n项和

程序接口如下：

2.代码实现

```
#include <bits/stdc++.h>
using namespace std;
int m;
// 计算第n个斐波那契数
```

```
long long fibonacci_nth(int n)
{
    if(n==0)
        return 0;
    if(n==1 || n==2)
        return 1;
    return fibonacci_nth(n-1)+fibonacci_nth(n-2);
};
// 输出前n项斐波那契数列
void fibonacci_sequence(int n)
{
    for(int  i = 0;i < n; i++)
    {
        cout<<fibonacci_nth(i)<<" ";
        if(i!=0  && i%8==7)
            cout<<endl;
    }
};
// 计算前n项斐波那契数列的和
long long fibonacci_sum(int n)
{
    long long cnt = 0;
    for(int i=0;i < n;i++)
    {
        cnt=cnt+fibonacci_nth(i);
    }
    return cnt;
};

int main()
{
    cin>>m;
    fibonacci_nth(m);
    fibonacci_sequence(m);
    cout<<endl<<fibonacci_sum(m);
    return 0;
}
```

3. 功能说明

- **fibonacci_nth 函数**：采用递归方式计算第 n 个斐波那契数，通过边界条件（n=0 返回 0，n=1 或 n=2 返回 1）避免递归无限循环，适用于中小规模 n 的计算。
- **fibonacci_sequence 函数**：循环调用 fibonacci_nth 函数，输出前 n 项斐波那契数列，并设置每 8 项自动换行，提升输出结果的可读性。
- **fibonacci_sum 函数**：通过循环累加前 n 项斐波那契数的值，返回总和，采用 long long 类型避免数据溢出问题。

二、简单的排序问题

1.题目要求

请你编写一个C/C++程序，能够实现以下功能：

1. 生成一个包含10个随机整数的数组（随机数范围：0~99）
2. 输出排序前的原始数组
3. 使用至少两种不同的排序算法（）对数组进行升序排序
4. 分别输出每种排序算法的结果

程序接口如下：

```
// 生成随机数组
void generateArray(int arr[], int n);

// 打印数组
void printArray(int arr[], int n);

// 排序算法函数（至少实现两个，不局限于所给四种排序方式）
void bubbleSort(int arr[], int n);        // 冒泡排序
void selectionSort(int arr[], int n);     // 选择排序
void insertionSort(int arr[], int n);     // 插入排序
void quickSort(int arr[], int low, int high); // 快速排序
```

输出格式

```
原始数组: 12 45 3 78 23 56 89 43 67 1
冒泡排序: 1 3 12 23 43 45 56 67 78 89
选择排序: 1 3 12 23 43 45 56 67 78 89
```

一些小提示tips

随机数生成

1. 使用 rand() 函数生成随机数；
2. 通过 srand(time(0)) 设置随机种子；
3. 需包含头文件：<stdlib.h> 和 <time.h>。

排序算法

- 冒泡排序： 相邻元素比较交换
- 选择排序： 每次选择最小元素放到前面
- 插入排序 将元素插入到已排序序列的正确位置
- 快速排序 分治思想，选择基准元素

代码规范要求

1. 将每个排序算法封装为函数；

2. 主函数清晰调用各个算法；
3. 添加必要的注释说明。

2. 代码实现

```
#include <bits/stdc++.h>
using namespace std;
// 生成随机数组
int arr[1000];
void generateArray(int arr[], int n)
{
    for(int i=1;i<=n;i++)
    {
        arr[i]=rand()%100;
    }
};
// 打印数组
void printArray(int arr[], int n)
{
    cout<<"原始数组：";
    for(int  i=1;i<=n;i++)
    {
        cout<<arr[i]<<" ";
    }
    cout<<endl;
};
// 排序算法函数（至少实现两个，不局限于所给四种排序方式）
void bubbleSort(int arr[], int n)
{
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n-i;j++)
        {
            if(arr[j]>arr[j+1])
            {
                swap(arr[j],arr[j+1]);//这里可以改成引入temp三行代码交换
            }
        }
    }
    cout<<"冒泡排序：";
    for(int i=1;i<=n;i++)
        cout<<arr[i]<<" ";
    cout<<endl;
}
; // 冒泡排序

void selectionSort(int arr[], int n)
{
    int minn=0;
    for (int i = 1; i <= n; i++)
    {
        minn = i;
```

```
        for (int j = i+1; j <= n; j++)
        {
            if (arr[j] < arr[minn])
            {
                minn = j;
            }
        }
        swap(arr[i],arr[minn]);
    }
    cout<<"选择排序: ";
    for (int i = 1; i <= n; i++)
        cout<<arr[i]<<" ";
    cout<<endl;
}
; // 选择排序

void insertionSort(int arr[], int n)
{
    for(int i=1;i<=n;i++)
    {
        if(arr[i]<arr[i-1])
        {
            int temp = arr[i];
            int j;
            for(j=i-1; j>=0 && arr[j]>temp ;j--)
            {
                arr[j+1]=arr[j];
            }
            arr[j+1] = temp;
        }

    }
    cout<<"插入排序: ";
    for (int i = 1; i <= n; i++)
        cout<<arr[i]<<" ";
    cout<<endl;
}
; // 插入排序

void quickSort(int arr[], int low, int high)
{
    int i = low;
    int j = high;
    if(i >= j) {
        return;
    }
    int temp = arr[low];
    while(i != j) {
        while(arr[j] >= temp && i < j)
        {
            j--;
        }
    while(arr[i] <= temp && i < j)
    {
```

```
            i++;
        }
    if(i < j)
    {
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[low], arr[i]);
    quickSort(arr, low, i - 1);
    quickSort(arr, i + 1, high);
}
; // 快速排序

int main()
{
    generateArray(arr,10);
    printArray(arr,10);
    bubbleSort(arr,10);
    selectionSort(arr,10);
    insertionSort(arr,10);
    quickSort(arr,1,10);
    cout<<"快速排序：";
    for (int i = 1; i <= 10; i++)
        cout<<arr[i]<<" ";
    return 0;
}
```

3. 算法特点说明

排序算法	时间复杂度（平均）	空间复杂度	核心特点
冒泡排序	$O(n^2)$	O(1)	稳定排序，通过相邻元素交换，每轮将最大元素 “冒泡” 到末尾
选择排序	$O(n^2)$	O(1)	不稳定排序，每轮选择最小元素与当前位置交换，减少交换次数
插入排序	$O(n^2)$	O(1)	稳定排序，适用于小规模或接近有序数组，插入过程类似整理扑克牌
快速排序	O(nlogn)	O(logn)	不稳定排序，分治思想，平均效率高，是实际应用中常用的排序算法

三、学生成绩管理系统（拔高）

1.题目描述

请你完成一个学生成绩管理系统的程序，要求实现以下功能：

1. 输入多个学生的成绩

2. 计算所有学生的平均分
3. 查找最高分和最低分
4. 对成绩进行降序排序
5. 统计各分数段人数

功能要求

输入学生成绩

- 从键盘输入学生人数
- 依次输入每个学生的成绩（0-100分）
- 使用指针遍历数组进行输入

计算平均分

- 编写函数计算所有成绩的平均值
- 返回数据类型为double的平均分

查找最高分和最低分

- 使用指针在数组中查找最大值和最小值
- 通过指针参数返回结果

成绩排序

- 对成绩数组进行降序排序
- 使用指针操作数组元素

统计等级

- 按以下等级统计人数： 优秀：90-100分 良好：80-89分 中等：70-79分 及格：60-69分 不及格：0-59分

函数接口

```
/*
  输入学生成绩
  scores 指向成绩数组的指针
  n 学生人数
 */
void inputScores(int *scores, int n);

/*
 计算平均分
 scores 指向成绩数组的指针
 n 学生人数
 return 平均分
 */
double calculateAverage(int *scores, int n);
```

```c
/*
 查找最高分和最低分
 scores 指向成绩数组的指针
 n 学生人数
 max 指向存储最高分的变量的指针
 min 指向存储最低分的变量的指针
 */
void findMinMax(int *scores, int n, int *max, int *min);

/*
 对成绩进行降序排序
 scores 指向成绩数组的指针
 n 学生人数
 */
void sortScores(int *scores, int n);

/*
 统计各等级人数
 scores 指向成绩数组的指针
 n 学生人数
 counts 指向等级统计数组的指针
 counts[0]: 优秀人数(90-100)
 counts[1]: 良好人数(80-89)
 counts[2]: 中等人数(70-79)
 counts[3]: 及格人数(60-69)
 counts[4]: 不及格人数(0-59)
 */
void countGrades(int *scores, int n, int *counts);
```

主函数框架

```c
int main() {
    int numStudents;

    printf("请输入学生人数: ");
    scanf("%d", &numStudents);

    int scores[numStudents];
    int gradeCounts[5] = {0};

    // 调用各功能函数
    inputScores(scores, numStudents);

    double avg = calculateAverage(scores, numStudents);
    printf("平均分: %.2f\n", avg);

    int maxScore, minScore;
    findMinMax(scores, numStudents, &maxScore, &minScore);
    printf("最高分: %d, 最低分: %d\n", maxScore, minScore);
```

```
    sortScores(scores, numStudents);
    printf("成绩降序排列: ");
    for(int i = 0; i < numStudents; i++) {
        printf("%d ", scores[i]);
    }
    printf("\n");

    countGrades(scores, numStudents, gradeCounts);
    printf("等级统计:\n");
    printf("优秀(90-100): %d人\n", gradeCounts[0]);
    printf("良好(80-89): %d人\n", gradeCounts[1]);
    printf("中等(70-79): %d人\n", gradeCounts[2]);
    printf("及格(60-69): %d人\n", gradeCounts[3]);
    printf("不及格(0-59): %d人\n", gradeCounts[4]);

    return 0;
}
```

运行Test（可测试多组数据）

输入示例：

```
请输入学生人数: 5
请输入5个学生的成绩:
85
92
78
65
88
```

输出样例：

```
平均分: 81.60
最高分: 92, 最低分: 65
成绩降序排列: 92 88 85 78 65
等级统计:
优秀(90-100): 1人
良好(80-89): 2人
中等(70-79): 1人
及格(60-69): 1人
不及格(0-59): 0人
```

2. 代码实现

```
#include <bits/stdc++.h>
using namespace std;
void inputScores(int *scores, int n) {
    printf("请输入%d个学生的成绩:\n", n);
    for (int i = 0; i < n; i++)
    {
        cin>>scores[i];
    }
}

double calculateAverage(int *scores, int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += scores[i];
    }
    return (double)sum / n;
}

void findMinMax(int *scores, int n, int *max, int *min) {
    *max = scores[0];
    *min = scores[0];
    for (int i = 1; i < n; i++) {
        if (scores[i] > *max) {
            *max = scores[i];
        }
        if (scores[i] < *min) {
            *min = scores[i];
        }
    }
}

void sortScores(int *scores, int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (scores[i] < scores[j]) {
                swap(scores[i],scores[j]);
            }
        }
    }
}

void countGrades(int *scores, int n, int *counts) {
    for (int i = 0; i < n; i++) {
        if (scores[i] >= 90) {
            counts[0]++;
        } else if (scores[i] >= 80) {
            counts[1]++;
        } else if (scores[i] >= 70) {
            counts[2]++;
        } else if (scores[i] >= 60) {
            counts[3]++;
        } else {
            counts[4]++;
```

```
            }
        }
    }

    int main() {
        int numStudents;
        printf("请输入学生人数: ");
        scanf("%d", &numStudents);

        int scores[numStudents];
        int gradeCounts[5] = {0};

        inputScores(scores, numStudents);

        double avg = calculateAverage(scores, numStudents);
        printf("平均分: %.2f\n", avg);

        int maxScore, minScore;
        findMinMax(scores, numStudents, &maxScore, &minScore);
        printf("最高分: %d, 最低分: %d\n", maxScore, minScore);

        sortScores(scores, numStudents);
        printf("成绩降序排列: ");
        for (int i = 0; i < numStudents; i++) {
            printf("%d ", scores[i]);
        }
        printf("\n");

        countGrades(scores, numStudents, gradeCounts);
        printf("等级统计:\n");
        printf("优秀(90-100): %d人\n", gradeCounts[0]);
        printf("良好(80-89): %d人\n", gradeCounts[1]);
        printf("中等(70-79): %d人\n", gradeCounts[2]);
        printf("及格(60-69): %d人\n", gradeCounts[3]);
        printf("不及格(0-59): %d人\n", gradeCounts[4]);

        return 0;
    }
```

3. 功能模块说明

- **输入模块（inputScores 函数）**：接收用户输入的学生人数和对应成绩，将成绩存储到数组中，支持灵活的人数设置。
- **统计模块**：包含 calculateAverage（计算平均分，保留 2 位小数）、findMinMax（遍历数组查找最高分和最低分）、countGrades（按分数段统计各等级人数）三个子功能，全面覆盖成绩统计需求。
- **排序模块（sortScores 函数）**：采用简单选择排序思想实现成绩降序排列，方便直观查看成绩排名。

三、作业总结

本次作业完成了三个核心编程任务，各任务功能实现完整，测试验证通过，具体总结如下：

1. **斐波那契数列任务**：使用递归方法实现了 “计算第 n 个斐波那契数”“输出前 n 项斐波那契数列”“计算前 n 项斐波那契数列和” 三个功能，考虑了数据溢出问题（采用 long long 类型）和输出可读性（每 8 项换行）。

2. **排序算法任务**：实现了冒泡排序、选择排序、插入排序（时间复杂度均为 O (n^2)）和快速排序（时间复杂度 O (nlogn)）四种经典算法，通过随机数组生成和结果输出，清晰展示了各算法的排序效果，对比了不同算法的特点。

3. **学生成绩管理系统任务**：围绕学生成绩管理场景，实现了成绩输入、平均分计算、最高分 / 最低分查找、成绩降序排序、等级统计五大功能，模块划分清晰，交互友好，满足基础成绩管理需求。