# 实验二：函数与数组（题目2、3、4、5）

## 题目 2 函数的递归调用和多文件结构

### 实验描述：

编写一个函数，求从 n个不同的数中取r 个数的所有选择的个数。其个数值为：其中： n! = n \* (n-1) \* (n-2) \* ... \* 1。

### 实验代码：

/\*

 \* @FILE: 2digui.cpp

 \* @Description: 编写一个函数，求从 n 个不同的数中取r 个数的所有选择的个数。

 \*/

// g++ -o 2digui 2digui.cpp func.cpp

#include <iostream>

#include "func.h"

using namespace std;

int main()

{

    int n, r;

    while (true)

    {

        cout << "Input n and r:";

        cin >> n >> r;

        if (n == 0 && r == 0) // 输入0 0结束程序

            break;

        if (n < r || n < 1 || r < 1) // 输入错误提示

        {

            cout << "Input error" << endl;

            continue;

        }

        double result = Cnr(n, r); // 调用func.cpp中的Cnr函数

        cout << n << " " << r << " " << result << endl;

    }

    return 0;

}

/\*

 \* @FILE: func.h

 \* @Description: func.cpp头文件，用于声明fn和Cnr函数

 \*/

#ifndef FUNC\_H

#define FUNC\_H

double fn(int n);

double Cnr(int n, int r);

#endif // FUNC\_H

/\*

 \* @FILE: func.cpp

 \* @Description: 实现func.h中的fn和Cnr函数

 \*/

#include "func.h"

// 计算 n!

double fn(int n)

{

    double result = 1;

    for(int i = 2; i <= n; i++)// 从2开始，计算到n

    {

        result \*= i;

    }

    return result;

}

// 计算从 n 个数中取 r 个数的组合数

double Cnr(int n, int r)

{

    double result = 1;

    for(int i = 1; i <= r; i++)// 从1开始，计算到r

    {

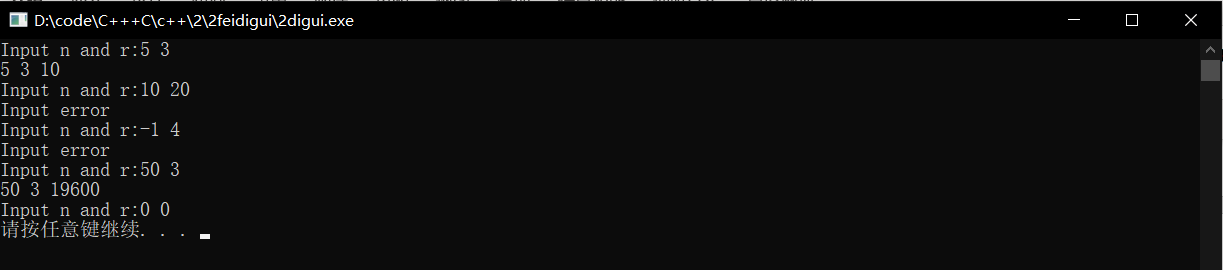
        result \*= (n - r + i) / static\_cast<double>(i);

    }

    return result;

}

### 实验结果：



### 思考与扩展:

1. 递归函数的书写要点是什么？

基本情况：递归函数必须有基本情况，这是终止递归的条件。当递归达到基本情况时，递归过程就会停止并返回结果。

递归情况：递归函数必须定义递归情况，也就是递归调用函数自身的情况。递归情况应该朝着基本情况靠近，否则会导致无限递归，最终导致栈溢出。

形参传递：递归函数应该定义形参来传递参数，这些参数随着递归调用而变化。通常情况下，形参包括当前计算的值和递归的深度。

返回值：递归函数往往需要返回一个值。在递归调用中，每个函数都应该返回一个值，这些返回值最终合并在一起，形成最终结果。

1. 在递归版本中，当n 大到什么值时系统将发生栈溢出报错？你觉得递归和非递归函数哪种好些？

2e9，非递归函数好一点，递归函数不但运行较为缓慢，且容易发生栈溢出。

1. 多文件结构中头文件的作用是什么？

在多文件结构中，头文件的作用是定义接口和声明函数、变量等信息。头文件通常包含函数、宏、数据结构等声明，提供给其他文件使用。头文件的主要作用包括：

1. 将程序划分为多个文件有什么好处？
   1. 提高代码的可读性和可维护性
   2. 降低编译时间和空间复杂度
   3. 支持代码重用
   4. 增强代码的可移植性

## 题目 3 排序查找

### 实验描述：

用选择法对 15 个学生的成绩按从大到小的顺序排序，15 个学生的成绩整数用cin 输入，如果输入的成绩不在[0，100]之间，则提示“输入错误”。排序完成后，输入一个成绩，要求用折半查找法找出该成绩是该组中第几个元素的值（即第几名）。如果该成绩不在数组中，则输出“无此成绩”。

要求：

1、把排序算法写成函数形式，在主函数中输入15 个数据，然后调用排序函数排序。

2、在排序过程中尽量减少数据的交换和移动。

3、把查找算法写成函数形式，在主函数中输入1 个数据，然后调用查找函数排序。

### 实验代码：

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\2\3sort.cpp

 \* @Description: 选择排序和折半查找

 \*/

#include <iostream>

using namespace std;

/\*\*

 \* @description: 实现选择排序

 \* @param {int} arr 数组

 \* @param {int} n   数组长度

 \* @return {\*}

 \*/

void selectionSort(int arr[], int n)

{

    for (int i = 0; i < n - 1; i++) // 从0开始，到n-1

    {

        int minIndex = i;

        for (int j = i + 1; j < n; j++) // 从i+1开始，到n

        {

            if (arr[j] < arr[minIndex]) // 如果arr[j]比arr[minIndex]小

            {

                minIndex = j; // 更新minIndex

            }

        }

        swap(arr[i], arr[minIndex]);

    }

}

/\*\*

 \* @description: 实现折半查找

 \* @param {int} arr 折半查找的数组

 \* @param {int} n   数组长度

 \* @param {int} target  要查找的目标

 \* @return {\*} 返回目标的下标，如果没有找到，返回-1

 \*/

int binarySearch(int arr[], int n, int target)

{

    int left = 0, right = n - 1;

    while (left <= right) // 如果left大于right，说明没有找到

    {

        int mid = left + (right - left) / 2; // 防止溢出

        if (arr[mid] == target)              // 如果找到了

        {

            return mid; // 返回下标

        }

        else if (arr[mid] < target) // 如果目标比中间值大

        {

            left = mid + 1; // 更新left

        }

        else

        {

            right = mid - 1; // 更新right

        }

    }

    return -1;

}

int main()

{

    const int n = 15;

    int arr[n];

    int score;

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

        if (arr[i] < 0 || arr[i] > 100) // 如果输入的数据不在0到100之间

        {

            cout << "Input error, the input data should be between 0 and 100" << endl;

            i--;

            continue;

        }

    }

    selectionSort(arr, n);

    cout << "sorted array:";

    for (int i = 0; i < n; i++) // 输出排序后的数组

    {

        cout << arr[i] << " ";

    }

    cout << endl;

    cout << "Input the score you want to search:";

    cin >> score;

    int index = binarySearch(arr, n, score); // 查找score的下标

    if (index == -1)

    {

        cout << "Not found" << endl;

    }

    else

    {

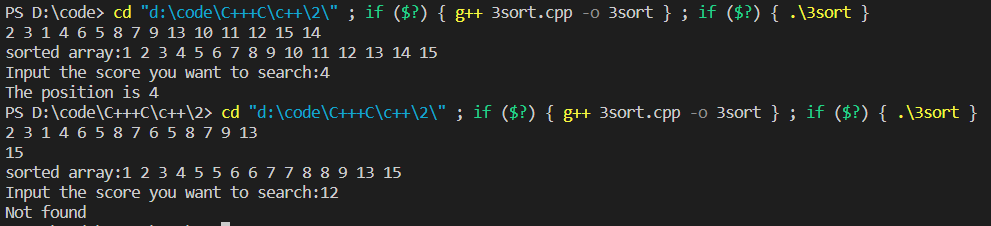
        cout << "The position is " << index + 1 << endl;

    }

    return 0;

}

### 实验结果：



### 思考与扩展：

1. 将选择法排序与起泡法比较，分析它们的特点和效率。

它们的主要区别在于选择排序是通过选择最小（或最大）元素来进行排序，而冒泡排序是通过相邻元素的比较和交换来进行排序。选择排序和冒泡排序的时间复杂度都是O(n^2)，但选择排序的性能稍微好一些，因为它在每一轮循环中只进行一次交换，而冒泡排序可能进行多次交换。然而，对于较小规模的数组，选择排序和冒泡排序的差异并不明显。

1. 如果采用顺序查找的办法，即从第一个元素开始对比一直到找到与之相等的元素，和折半查找法相比，那个效率高？

折半查找法的效率要比顺序查找法高。

在最坏的情况下，即目标元素位于列表的最后，顺序查找的时间复杂度为O(n)。折半查找通过不断将待查找范围缩小为一半来进行查找，每次比较目标元素与中间元素的大小，并根据比较结果确定下一步的查找范围。通过每次将查找范围缩小一半，折半查找的时间复杂度为O(log n)。

1. 数组名作为函数参数，形参和实参的关系？

形参是指针：函数参数声明中的数组形参实际上是一个指针变量，可以使用指针操作符(\*)来操作数组。在这种情况下，函数内部对形参数组的修改会直接反映到函数外部的实参数组上。例如：

void myFunction(int\* arr, int size) {

    // 对数组进行操作，可以使用arr[i]来访问数组元素

}

形参是数组：在函数参数声明中，可以直接声明一个数组形参，但实际上它会被自动转换为指向数组首元素的指针。在这种情况下，形参数组本质上是一个指针，函数内部对形参数组的修改同样会反映到函数外部的实参数组上。例如：

void myFunction(int arr[], int size) {

    // 对数组进行操作，可以使用arr[i]来访问数组元素

}

## 题目 4 矩阵相乘

### 实验描述：

将两个给定的距阵(3＊3)相乘得到另一个距阵并将其打印出来。

### 实验代码：

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\2\4matrix\_multiply.cpp

 \* @Description: 计算两个3\*3矩阵的乘积

 \*/

#include <iostream>

using namespace std;

/\*\*

 \* @description: 计算矩阵乘法

 \* @param {int} a 第一个矩阵A

 \* @param {int} b 第二个矩阵B

 \* @param {int} c 结果矩阵C

 \* @return {\*}

 \*/

void matrix\_multiply(int a[3][3], int b[3][3], int c[3][3])

{

    for (int i = 0; i < 3; i++) // i表示行

    {

        for (int j = 0; j < 3; j++) // j表示列

        {

            c[i][j] = 0; // 初始化

            for (int k = 0; k < 3; k++)

            {

                c[i][j] += a[i][k] \* b[k][j]; // 矩阵乘法

            }

        }

    }

}

int main()

{

    int a[3][3], b[3][3], c[3][3];

    cout << "Input A matrix(3\*3)" << endl;

    for (int i = 0; i < 3; i++)

    {

        for (int j = 0; j < 3; j++)

        {

            cin >> a[i][j];

        }

    }

    cout << "Input B matrix(3\*3)" << endl;

    for (int i = 0; i < 3; i++)

    {

        for (int j = 0; j < 3; j++)

        {

            cin >> b[i][j];

        }

    }

    matrix\_multiply(a, b, c);

    cout << "The answer of A\*B" << endl;

    for (int i = 0; i < 3; i++)

    {

        for (int j = 0; j < 3; j++)

        {

            cout << c[i][j] << " ";

        }

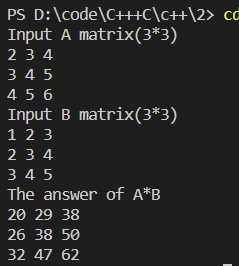
        cout << endl;

    }

    return 0;

}

### 实验结果：



### 思考与扩展：

1. 如果是一个4\*3 矩阵和一个3\*2 矩阵相乘，程序该如何改写？

关键代码改写如下：

void matrix\_multiply(int a[4][3], int b[3][2], int c[4][2])

{

    for (int i = 0; i < 4; i++) // i表示A矩阵的行

    {

        for (int j = 0; j < 2; j++) // j表示B矩阵的列

        {

            c[i][j] = 0; // 初始化

            for (int k = 0; k < 3; k++) // k表示A矩阵的列/B矩阵的行

            {

                c[i][j] += a[i][k] \* b[k][j]; // 矩阵乘法

            }

        }

    }

}

1. 多维数组名作为函数参数，形参和实参的关系？

形参是指针：多维数组名作为函数参数时，形参实际上是一个指向数组首元素的指针。无论是一维还是多维数组，都可以使用指针操作符(\*)来操作数组。在这种情况下，形参是一个指向多维数组的指针，函数内部可以使用多维数组的下标来访问数组元素。例如：

void myFunction(int (\*arr)[3], int size) {

    // 对数组进行操作，可以使用arr[i][j]来访问数组元素

}

形参是多维数组：在函数参数声明中，可以直接声明一个多维数组形参。在这种情况下，形参是一个多维数组，函数内部可以使用多维数组的下标来访问数组元素。例如：

void myFunction(int arr[][3], int size) {

    // 对数组进行操作，可以使用arr[i][j]来访问数组元素

}

需要注意的是，多维数组作为函数参数时，除了第一维的大小可以不指定外，其他维度的大小必须指定。在函数声明和定义中，我们需要明确指定多维数组的每个维度的大小，以便正确访问数组元素。

## 题目 5 字符串连接

### 实验描述：

编一程序，将两个字符串连接起来。结果取代第一个字符串。要求

1、用字符数组，不用strcat 函数；

2、用C 标准中的strcat 函数；

3、用string 方法定义字符串变量；

4、对这三种方法进行比较。

### 实验代码：

/\*

 \* @Author: 王贤义

 \* @FilePath: \code\C+++C\c++\2\5char.cpp

 \* @Description: 编一程序，将两个字符串连接起来。结果取代第一个字符串。要求

                1、用字符数组，不用strcat 函数；

                2、用C 标准中的strcat 函数；

                3、用string 方法定义字符串变量；

 \*/

#include <iostream>

#include <cstring>

using namespace std;

int main()

{

    // Method 1: Using character arrays

    char str1[100] = "Hello";

    char str2[] = "World";

    int i = 0, j = 0;

    while (str1[i] != '\0')

    {

        i++;

    }

    while (str2[j] != '\0')

    {

        str1[i] = str2[j];

        i++;

        j++;

    }

    str1[i] = '\0';

    // Method 2: Using strcat function

    char str3[100] = "Hello";

    char str4[] = "World";

    strcat(str3, str4);

    // Method 3: Using string method

    string str5 = "Hello";

    string str6 = "World";

    str5 += str6;

    // Comparing the three methods

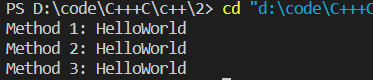
    cout << "Method 1: " << str1 << endl;

    cout << "Method 2: " << str3 << endl;

    cout << "Method 3: " << str5 << endl;

}

### 实验结果：



### 思考与扩展：

1. 字符数组和字符串数组有什么联系或区别？

字符数组和字符串数组都可以表示字符串，但字符串数组更符合字符串的规范，使用字符串处理函数更方便。字符数组可以表示任意的字符序列，不一定要符合字符串的规范，并且需要额外的处理来确定字符串的长度和进行字符串操作。