11月19日 程序设计数学公式类题目

1.编写程序,因素安调和剂数H,要求最贱分数形式 A/B输入,例如 n=5 时候,输出 137/60。 H=1+1/2+1/3+...+1/n

注意需要通分，输出的是分数的形式。

答案:

```c
#include <stdio.h>
// 求最大公约数
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int main() {
    int n;
    printf("请输入 n 的值: ");
    scanf("%d", &n);

    int numerator = 0;   // 分子
    int denominator = 1; // 分母
    // 输出最简分数形式
    for (int i = 1; i <= n; i++) {
        numerator = numerator * i + denominator; // 通分并加和
        denominator *= i;                        // 更新分母

        // 化简分数
        int g = gcd(numerator, denominator);
        numerator /= g;
        denominator /= g;
    }
    printf("%d/%d\n", numerator, denominator);
    return 0;
}
```

2.平面有 100个点，任意三个点可以构成一个三角形。编一个程序，输入 100个点的坐标，输出在构成的所有三角形中，最大的三角形的面积。

思路：

1. **三角形的面积计算**： 给定三个点 `(x1, y1)`，`(x2, y2)` 和 `(x3, y3)`，三角形的三个顶点坐标求其面积的公式为:

$$面积 = \frac{1}{2}|x1(y2 - y3) + x2(y3 - y1) + x3(y1 - y2)|$$

   这个公式通过计算矩阵的行列式得到一个平行四边形的面积，取绝对值后再除以2得到三角形的面积。

2. **暴力法**： 由于题目中给定了100个点，我们需要遍历所有三点组合（即 `C(100, 3)` = 161700 个三角形），对于每一组点使用上述公式计算三角形面积，并记录最大的面积。

3. **输入输出**:
   - 输入：100个点的坐标。
   - 输出：最大的三角形的面积。

代码：

```
#include <stdio.h>
#include <math.h>

// 定义一个结构体来表示二维点
typedef struct {
    double x, y;
} Point;

// 计算三点所形成的三角形的面积
double calculateArea(Point p1, Point p2, Point p3) {
    return 0.5 * fabs(p1.x * (p2.y - p3.y) + p2.x * (p3.y - p1.y) + p3.x * (p1.y
- p2.y));
}

int main() {
    Point points[100]; // 存储100个点的坐标
    double maxArea = 0; // 用于记录最大的三角形面积
    int i, j, k;

    // 输入100个点的坐标
    printf("请输入100个点的坐标（每个点的坐标格式为x y）：\n");
    for(i = 0; i < 100; i++) {
        scanf("%lf %lf", &points[i].x, &points[i].y);
    }

    // 遍历所有三点组合，计算三角形的面积
    for(i = 0; i < 100; i++) {
        for(j = i + 1; j < 100; j++) {
            for(k = j + 1; k < 100; k++) {
                double area = calculateArea(points[i], points[j], points[k]);
```

```
                if(area > maxArea) {
                    maxArea = area; // 更新最大面积
                }
            }
        }
    }

    // 输出最大面积
    printf("最大的三角形的面积是: %.2lf\n", maxArea);

    return 0;
}
```

3.求sinx近似值

$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$编写程序，求$sinx$的近似值，要求误差小于$10^{-8}$

算法思路：

1. 根据$sin(x)$的展开式：$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$
2. 每一项可以通过上一项递推得到：$\text{term}_{n+1} = -\text{term}_n \times \frac{x^2}{(2n+2)(2n+3)}$
3. 初始项设置为x，逐项累加，直到当前项的绝对值小于10^{-8}
4. 使用递推减少重复计算，优化性能。

答案：

```
#include <stdio.h>

// 函数计算 sin(x)
double computeSin(double x) {
    double term = x;         // 当前项的值
    double sum = term;       // 累计的结果
    double threshold = 1e-8; // 误差阈值
    int n = 1;               // 当前项的阶数（1 表示 x^1）

    // 逐项计算直到误差小于阈值
    while (1) {
        term *= -x * x / ((2 * n) * (2 * n + 1)); // 计算下一项
        if (term > -threshold && term < threshold) {
            break; // 如果当前项绝对值小于误差阈值，停止计算
        }
        sum += term; // 累加到结果
        n++;         // 更新阶数
    }
    return sum;
}
```

```c
int main() {
    double x;
    printf("请输入角度（弧度制）x: ");
    scanf("%lf", &x);

    double result = computeSin(x);
    printf("sin(%.6f) 的近似值为: %.8f\n", x, result);

    return 0;
}
```

4. 平面有100个点，任意两点可以构成一个线段。编一个程序，输出在构成的的所有线段中:长度最长的线段长度，两点(xl，y1)，(x2，y2)之间的距离公式为:

$$D = \sqrt{(x2 - x1)2 + (y2 - y1)2}$$

思路:

1. **计算两点之间的距离**： 通过上述公式，计算任意两点 `(x1, y1)` 和 `(x2, y2)` 之间的距离。该距离为欧几里得距离，可以通过平方差求和并取平方根得到。
2. **暴力法**： 我们需要遍历所有点对，计算每一对点的距离。对于所有的点对，找出最大的距离。
3. **输入输出**:
    - 输入：100个点的坐标。
    - 输出：所有点对中最长的线段的长度。

代码:

```c
#include <stdio.h>
#include <math.h>

// 定义一个结构体来表示二维点
typedef struct {
    double x, y;
} Point;

// 计算两点之间的距离
double calculateDistance(Point p1, Point p2) {
    return sqrt((p2.x - p1.x) * (p2.x - p1.x) + (p2.y - p1.y) * (p2.y - p1.y));
}

int main() {
    Point points[100]; // 存储100个点的坐标
    double maxDistance = 0; // 用于记录最大的线段长度
    int i, j;

    // 输入100个点的坐标
    printf("请输入100个点的坐标（每个点的坐标格式为x y）：\n");
    for(i = 0; i < 100; i++) {
```

```
        scanf("%lf %lf", &points[i].x, &points[i].y);
    }

    // 遍历所有点对，计算两点之间的距离并更新最大距离
    for(i = 0; i < 100; i++) {
        for(j = i + 1; j < 100; j++) {
            double distance = calculateDistance(points[i], points[j]);
            if(distance > maxDistance) {
                maxDistance = distance; // 更新最大距离
            }
        }
    }

    // 输出最长线段的长度
    printf("最长线段的长度是: %.2lf\n", maxDistance);

    return 0;
}
```

5.有100个正整数存放在数组中，试编一函数，要求:(I)把所有的偶数按从小到大的顺序放在数组的前半部。(2)把所有的奇数按从小到大的顺序放在数组的后半部。

例如: 143259 7
输出：2413579

思路：

1. **分离偶数和奇数**:
   - 创建两个数组：一个用于存放偶数，另一个用于存放奇数。
   - 遍历给定的数组，将偶数放入偶数数组，奇数放入奇数数组。
2. **排序**:
   - 对偶数数组进行升序排序。
   - 对奇数数组进行升序排序。
3. **合并**:
   - 将排序后的偶数数组和奇数数组合并到原数组中。偶数数组放在前面，奇数数组放在后面。

代码：

```
#include <stdio.h>

// 冒泡排序函数
void bubbleSort(int arr[], int n) {
    int temp;
    // 外层循环控制遍历次数
    for (int i = 0; i < n - 1; i++) {
        // 内层循环进行相邻元素比较
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
```

```
                // 交换相邻元素
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

void rearrange(int arr[], int n) {
    int evens[n], odds[n]; // 分别存储偶数和奇数
    int evenCount = 0, oddCount = 0;

    // 将偶数和奇数分开存放
    for (int i = 0; i < n; i++) {
        if (arr[i] % 2 == 0) {
            evens[evenCount++] = arr[i];
        } else {
            odds[oddCount++] = arr[i];
        }
    }

    // 对偶数和奇数分别进行冒泡排序
    bubbleSort(evens, evenCount);
    bubbleSort(odds, oddCount);

    // 将排序后的偶数和奇数合并回原数组
    int index = 0;
    for (int i = 0; i < evenCount; i++) {
        arr[index++] = evens[i];
    }
    for (int i = 0; i < oddCount; i++) {
        arr[index++] = odds[i];
    }
}

int main() {
    int arr[] = {1, 4, 3, 2, 5, 9, 7}; // 示例输入
    int n = sizeof(arr) / sizeof(arr[0]);

    // 调用函数重新排列
    rearrange(arr, n);

    // 输出排序后的数组
    printf("排序后的数组: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

6.给定一个正整数 N，由所有分母小于或等于的最简真分数按从小到大组成一个序列，例如，N=5　1/2 1/3 2/3 1/4 3/4 1/5 2/5 3/5 4/5编一个程序，输入一个正整数，输出上述序列.

思路：

计算最大公约数：使用 `gcd` 函数判断分子和分母是否互质。

遍历每个分母：对于每个分母，找出所有满足条件的分子。

使用冒泡排序：通过冒泡排序对分数进行排序。

输出结果：按从小到大的顺序输出所有最简真分数。

答案：

```c
#include <stdio.h>

// 计算最大公约数
int gcd(int a, int b) {
    while (b != 0) {
        int temp = a % b;
        a = b;
        b = temp;
    }
    return a;
}


// 打印排序后的分数
void printFractions(int N) {
    int fractions[1000][2];  // 用来存放所有的分子和分母
    int count = 0;  // 记录分数的个数

    // 遍历每个分母
    for (int denominator = 2; denominator <= N; denominator++) {
        for (int numerator = 1; numerator < denominator; numerator++) {
            if (gcd(numerator, denominator) == 1) {  // 判断分子和分母是否互质
                fractions[count][0] = numerator;
                fractions[count][1] = denominator;
                count++;
            }
        }
    }

    // 输出结果
    for (int i = 0; i < count; i++) {
        printf("%d/%d ", fractions[i][0], fractions[i][1]);
    }
    printf("\n");
}

int main() {
```

```
    int N;
    printf("请输入一个正整数 N: ");
    scanf("%d", &N);

    // 打印分数
    printFractions(N);

    return 0;
}
```

输入：

```
请输入一个正整数 N: 5
```

输出：

```
1/2 1/3 2/3 1/4 3/4 1/5 2/5 3/5 4/5
```

7. 进行高精度计算，我们可以用一个数组表示一个正整数，一个数组元素表示整数的一位，例如 396 可以用数组A表示,即 A[1]=6,A[2]=9A[3]=3，编一个函数，计算这样表示的两个整数A,B之积，积存放在数组C中。注:假定积不会超过 100 位。

思路：

1. **输入表示**：两个整数 `A` 和 `B` 用数组表示，每个数组的元素代表该整数的每一位。假设整数的最高位在数组的最后一个元素。
2. **逐位乘法**：对于两个数字的每一位，执行逐位乘法，并且考虑到进位。每次计算乘积时将结果存储到一个数组 `C` 中。
3. **进位处理**：需要将每一位的结果加到相应的位置，并处理进位。
4. **输出**：最终的结果存储在数组 `C` 中，输出结果时从数组的高位到低位输出。

代码

```
#include <stdio.h>
#include <string.h>


// 计算高精度大数相乘
void multiply(int A[], int B[], int C[], int lenA, int lenB) {
    // 初始化结果数组 C，所有位置都为0
    for (int i = 0; i < lenA + lenB; i++) {
        C[i] = 0;
    }

    // 逐位相乘
    for (int i = 0; i < lenA; i++) {
```

```
        for (int j = 0; j < lenB; j++) {
            C[i + j] += A[i] * B[j];  // 乘法累加到对应的位置
            // 处理进位
            if (C[i + j] >= 10) {
                C[i + j + 1] += C[i + j] / 10;
                C[i + j] %= 10;
            }
        }
    }
}

// 打印结果
void printResult(int C[], int lenC) {
    int start = lenC - 1;
    // 找到第一个非零的位数
    while (start >= 0 && C[start] == 0) {
        start--;
    }

    if (start == -1) {  // 如果没有有效位，则结果是 0
        printf("0");
    } else {
        for (int i = start; i >= 0; i--) {
            printf("%d", C[i]);
        }
    }
    printf("\n");
}

int main() {
    char strA[100 + 1], strB[100 + 1];
    int A[100], B[100], C[2 * 100] = {0};
    int lenA, lenB;

    // 输入两个大整数
    printf("请输入第一个大整数：");
    scanf("%s", strA);
    printf("请输入第二个大整数：");
    scanf("%s", strB);

    // 将输入的数字字符串转化为数组
    lenA = strlen(strA);
    lenB = strlen(strB);

    for (int i = 0; i < lenA; i++) {
        A[i] = strA[lenA - 1 - i] - '0';  // 从低位到高位存储
    }

    for (int i = 0; i < lenB; i++) {
        B[i] = strB[lenB - 1 - i] - '0';  // 从低位到高位存储
    }

    // 调用乘法函数
    multiply(A, B, C, lenA, lenB);

    // 输出结果
```

```
    printf("结果为：");
    printResult(C, lenA + lenB);
    return 0;
}
```

8.任意一个大于2的偶数，都可以分解为两个质数之和。编写一个程序，验证上述结论。 例如:输入16，输出 16=13+3

思路：

1. 遍历从2到输入偶数的一半的数字，判断它们是否为质数。
2. 对于每一个质数 `p1`，计算 `p2 = N - p1`，如果 `p2` 也是质数，输出结果 `N = p1 + p2`

代码：

```
#include <stdio.h>

// 判断一个数是否是质数
bool isPrime(int n) {
    if (n <= 1) return false;  // 1 不是质数
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) return false;  // 如果能被整除，说明不是质数
    }
    return true;
}

// 主程序：验证哥德巴赫猜想
void verifyGoldbachConjecture(int n) {
    // 寻找符合条件的两个质数
    for (int p1 = 2; p1 <= n / 2; p1++) {
        if (isPrime(p1)) {
            int p2 = n - p1;
            if (isPrime(p2)) {
                printf("%d = %d + %d\n", n, p1, p2);
                return;
            }
        }
    }
}

int main() {
    int number;

    // 输入一个偶数
    printf("请输入一个大于2的偶数：");
    scanf("%d", &number);

    // 调用函数验证哥德巴赫猜想
    verifyGoldbachConjecture(number);

    return 0;
}
```

9.(3x+5y)的n次幂的二项式展开有n+1项，按照x的指数升序排列，编写递归函数，返回该展开式的第 k(0<=k<=n)项的系数，递归函数声明参考形式为 int find (int n,int k). 例:(3x+5y)3=125y3+225xy2+135x2y+27x，则其第二项系数为135(从0开始计数)说明:不允许使用数组和全局变量，不需考虑计算过程中类型溢出问题。

思路:

1. 组合数的递归形式：$C(n,k) = \frac{n!}{k!(n-k)!} = C(n-1,k-1) + C(n-1,k)$

2. **幂的计算**：我们可以递归地计算幂，函数 `power(a, b)` 用来计算 a^b。
3. **递归计算系数**：在递归计算过程中，我们需要不断计算组合数和相应的幂次。

代码

```
#include <stdio.h>

// 递归计算组合数 C(n, k)
int C(int n, int k) {
    if (k == 0 || k == n) {
        return 1;  // C(n, 0) = 1 和 C(n, n) = 1
    }
    return C(n - 1, k - 1) + C(n - 1, k);  // 递归计算 C(n, k)
}

// 递归计算 a^b
int power(int a, int b) {
    if (b == 0) {
        return 1;  // 任意数的0次幂为1
    }
    return a * power(a, b - 1);  // 递归计算 a^b
}

// 递归计算 (3x + 5y)^n 的第 k 项的系数
int find(int n, int k) {
    // 计算组合数 C(n, k)
    int comb = C(n, k);
    // 计算该项的系数
    int coefficient = comb * power(3, n - k) * power(5, k);
    return coefficient;
}

int main() {
```

```
    int n, k;

    // 输入n和k
    printf("请输入 n 和 k (n >= k >= 0): ");
    scanf("%d %d", &n, &k);

    // 查找并输出该项系数
    int result = find(n, k);
    printf("二项式展开式中第 %d 项的系数为: %d\n", k, result);

    return 0;
}
```