结构体

2022 年 8 月 16 日

1. 编写程序，使得用户输入两个分数 f_1 和 f_2，计算机返回 $(f_1^2+f_1\cdot f_2+f_2^2)/(f_1^2-f_1\cdot f_2+f_2^2)$ 的值。（注意，程序中不允许使用全局变量，包括全局基本类型变量、全局数组变量、全局指针变量和全局结构体变量）

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

struct Fraction{
    int numerator;
    int dominator;
    int sign; // 1 represents pos, 0 represents neg
};

struct Fraction simplified_version(struct Fraction f)
{
    // adjust the sign
    if(f.numerator < 0 && f.dominator < 0)
    {
        f.numerator = -f.numerator;
        f.dominator = -f.dominator;
    }
    else if(f.numerator < 0)
    {
        f.numerator = -f.numerator;
        f.sign = 1 - f.sign; // flip the sign
    }
    else if(f.dominator < 0)
    {
        f.dominator = -f.dominator;
```

```
26          f.sign = 1 - f.sign;
27      }
28      // here both the dominator and the numerator are positive
29      // reductions
30      int min = f.numerator > f.dominator ? f.dominator : f.numerator;
31      for(int i = 2; i <= min; i++)
32      {
33          if(f.numerator % i == 0 && f.dominator % i == 0)
34          {
35              f.numerator /= i;
36              f.dominator /= i;
37              i--;
38          }
39      }
40      return f;
41  }
42
43  struct Fraction sum(struct Fraction f1, struct Fraction f2)
44  {
45      struct Fraction fractionSum;
46      f1 = simplified_version(f1);
47      f2 = simplified_version(f2);
48      if(!f1.sign)
49      {
50          f1.sign = 1 - f1.sign;
51          f1.numerator = -f1.numerator;
52      }
53      if(!f2.sign)
54      {
55          f2.sign = 1 - f2.sign;
56          f2.numerator = -f2.numerator;
57      }
58      fractionSum.numerator = f1.numerator * f2.dominator + f2.numerator *
            f1.dominator;
59      fractionSum.dominator = f1.dominator * f2.dominator;
60      fractionSum.sign = 1;
61      return simplified_version(fractionSum);
62  }
63
```

```
struct Fraction negation(struct Fraction f)
{
    f = simplified_version(f);
    f.sign = 1 - f.sign;
    return f;
}

struct Fraction product(struct Fraction f1, struct Fraction f2)
{
    struct Fraction fractionProduct;
    f1 = simplified_version(f1);
    f2 = simplified_version(f2);
    if(!f1.sign)
    {
        f1.sign = 1 - f1.sign;
        f1.numerator = -f1.numerator;
    }
    if(!f2.sign)
    {
        f2.sign = 1 - f2.sign;
        f2.numerator = -f2.numerator;
    }
    fractionProduct.numerator = f1.numerator * f2.numerator;
    fractionProduct.dominator = f1.dominator * f2.dominator;
    fractionProduct.sign = 1;
    return simplified_version(fractionProduct);
}

struct Fraction inversion(struct Fraction f)
{
    f = simplified_version(f);
    struct Fraction inverseF;
    inverseF.numerator = f.dominator;
    inverseF.dominator = f.numerator;
    inverseF.sign = f.sign;
    return inverseF;
}


```

```
103 struct Fraction funct1(struct Fraction f1, struct Fraction f2)
104 {
105     struct Fraction part1 = product(f1, f1);
106     struct Fraction part2 = product(f1, f2);
107     struct Fraction part3 = product(f2, f2);
108     return sum(sum(part1, part2), part3);
109 }
110
111 struct Fraction funct2(struct Fraction f1, struct Fraction f2)
112 {
113     struct Fraction part1 = product(f1, f1);
114     struct Fraction part2 = negation(product(f1, f2));
115     struct Fraction part3 = product(f2, f2);
116     return sum(sum(part1, part2), part3);
117 };
118
119
120 struct Fraction fun(struct Fraction f1, struct Fraction f2)
121 {
122     return product(funct1(f1, f2), inversion(funct2(f1, f2)));
123 }
124
125 void input_fraction(struct Fraction *ptr_to_fraction)
126 {
127     printf("input the sign for a fractional number (1 for positive and 0 for
          negative):\n");
128     scanf("%d", &(ptr_to_fraction->sign));
129     if(ptr_to_fraction->sign != 1 && ptr_to_fraction->sign != 0)
130     {
131         printf("did not input 0 or 1\n");
132         exit(1);
133     }
134     printf("input the numerator:\n");
135     scanf("%d", &(ptr_to_fraction->numerator));
136     printf("input the dominator:\n");
137     scanf("%d", &(ptr_to_fraction->dominator));
138 }
139
140 void print_fraction(struct Fraction f)
```

```
{
    if(!f.sign) printf("-");
    printf("%d %d\n", f.numerator, f.dominator);
}

int main()
/*users input two fractional numbers: f1 and f2,
compute the expression (f1^2+f1*f2+f2^2)/(f1^2-f1*f2+f2^2)*/
{
    struct Fraction f1, f2;
    input_fraction(&f1);
    input_fraction(&f2);
    printf("The input fractional numbers are: \n");
    print_fraction(f1); print_fraction(f2);
    struct Fraction f3 = fun(f1, f2);
    printf("the result is:\n");
    print_fraction(f3);
    return 0;
}
```

---

2. 设 $\vec{a}=(x_a,y_a,z_a)$ 和 $\vec{b}=(x_b,y_b,z_b)$ 均为 3 维向量，定义

   - $\vec{a}+\vec{b}=(x_a+x_b,y_a+y_b,z_a+z_b)$；
   - $\vec{a}-\vec{b}=\vec{a}+(-\vec{b})$；
   - $\vec{a}\cdot\vec{b}=x_ax_b+y_ay_b+z_az_b$；
   - $\vec{a}\times\vec{b}=(y_az_b-y_bz_a,z_ax_b-x_az_b,x_ay_b-x_by_a)$；
   - $\lambda\vec{a}=(\lambda x_a,\lambda x_b,\lambda z_a)$，其中 λ 为实数。

   输入 $\vec{a}$，$\vec{b}$，$\vec{c}$ 的值，求 $(\vec{a}+\vec{b}-\vec{c})\times(\vec{a}-\vec{b}+\vec{c})\cdot(2\vec{a}-\vec{b}-\vec{c})$ 的值。

```
#include <stdio.h>

// 一个向量包含三个分量
struct Vector{
    double x;
    double y;
    double z;
};
```

```
9
10 struct Vector add(struct Vector va, struct Vector vb)
11 {
12     struct Vector v_sum; // C语言中，要带上struct；C++中，它可省
13     v_sum.x = va.x + vb.x; // va.x表示向量va的分量x
14     v_sum.y = va.y + vb.y;
15     v_sum.z = va.z + vb.z;
16     return v_sum; // 返回一个向量
17 }
18
19 struct Vector negation(struct Vector va)
20 {
21     struct Vector v_neg;
22     v_neg.x = -va.x;
23     v_neg.y = -va.y;
24     v_neg.z = -va.z;
25     return v_neg;
26 }
27
28 struct Vector minus(struct Vector va, struct Vector vb)
29 {
30     struct Vector v_diff;
31     return add(va, negation(vb));
32 }
33
34 double inner_product(struct Vector va, struct Vector vb)
35 {
36     return va.x * vb.x + va.y * vb.y + va.z * vb.z;
37 }
38
39 struct Vector cross_product(struct Vector va, struct Vector vb)
40 {
41     struct Vector v_cross_p;
42     v_cross_p.x = va.y * vb.z - vb.y * va.z;
43     v_cross_p.y = va.z * vb.x - va.x * vb.z;
44     v_cross_p.z = va.x * vb.y - vb.x * va.y;
45     return v_cross_p;
46 }
47
```

```c
struct Vector multiply(double lambda, struct Vector v)
{
    struct Vector result;
    result.x = lambda * v.x;
    result.y = lambda * v.y;
    result.z = lambda * v.z;
    return result;
}

void input_vector(struct Vector* p) // 传入指针以利于修改向量
{
    printf("input the components of a vector:\n");
    scanf("%lf%lf%lf", &(p->x), &(p->y), &(p->z));
}

void print_vector(struct Vector v)
{
    printf("(%lf, %lf, %lf)", v.x, v.y, v.z);
}

int main()
{
    struct Vector va, vb, vc;
    printf("input va:\n");
    input_vector(&va); // 传入地址以利于修改它
    printf("input vb:\n");
    input_vector(&vb);
    printf("input vc:\n");
    input_vector(&vc);

    printf("the input vectors are:\n");
    print_vector(va); printf("\n");
    print_vector(vb); printf("\n");
    print_vector(vc); printf("\n");

    struct Vector result1 = minus(add(va, vb), vc);
    struct Vector result2 = add(minus(va, vb), vc);
    struct Vector result3 = minus(minus(multiply(2, va), vb), vc);

```

```
87      double result = inner_product(cross_product(result1, result2), result3);
88      printf("result: %lf\n", result);
89
90      return 0;
91  }
```

3.