Exercise 4

Introduction to Computational Astrophysics, SoSe 2024

Seha Lee

Task 1. *Implement a complex data type*

Solution.

```cpp
#include <iostream>
using namespace std;

class complex_d {
public:
    double re, im;
    complex_d(double re, double im) : re(re), im(im) {}
    complex_d operator+(const complex_d& c) const {
        return complex_d(re + c.re, im + c.im);
    }
    complex_d operator-(const complex_d& c) const {
        return complex_d(re - c.re, im - c.im);
    }
    complex_d operator*(const complex_d& c) const {
        return complex_d(re * c.re - im * c.im, re * c.im + im * c.re);
    }
    complex_d operator/(const complex_d& c) const {
        double den = c.re * c.re + c.im * c.im;
        return complex_d((re * c.re + im * c.im) / den, (im * c.re -
            re * c.im) / den);
    }

    double absolute() const {
        return Heron_sqrt(re * re + im * im);
    }
    double Heron_sqrt(double a) const {
        double x = a;
        for (int i = 0; i < 10; i++) {
            x = 0.5 * (x + a / x);
        }
        return x;
    }
```

```
};


int main(){
   //four integer numbers from input
   double a, b, c, d;
   cout << "Enter four integers, separated by space: ";
   cin >> a >> b >> c >> d;
   //print as complex number
   complex_d c1(a, b);
   complex_d c2(c, d);
   cout << "c1 = " << c1.re << " + " << c1.im << "i" << endl;

   cout << "c2 = " << c2.re << " + " << c2.im << "i" << endl;

   //arithmetics
   complex_d sum = c1 + c2;
   complex_d diff = c1 - c2;
   complex_d prod = c1 * c2;
   complex_d quot = c1 / c2;
   double abs_c1 = c1.absolute();
   double abs_c2 = c2.absolute();

   //print results
   cout << "c1 + c2 = " << sum.re << " + " << sum.im << "i" << endl;
   cout << "c1 - c2 = " << diff.re << " + " << diff.im << "i" <<
      endl;
   cout << "c1 * c2 = " << prod.re << " + " << prod.im << "i" <<
      endl;
   cout << "c1 / c2 = " << quot.re << " + " << quot.im << "i" <<
      endl;
   cout << "|c1| = " << abs_c1 << endl;
   cout << "|c2| = " << abs_c2 << endl;
   return 0;
}
```

□

Task 2. *Absorption*

Solution. (a)

```
 float y = 100000010.0f;
  int start = 100000001, end = static_cast<int>(y);

  for (int i = start; i <= end; ++i) {
     float x = static_cast<float>(i);
  }
```

(b)
inc = 1.E-7 was added 1.E+7 times to yield the answer of 8. □

Method	Value
Regular summation (float)	7.0000000000
Kahan summation (float)	8.0000000000
Regular summation (double)	8.0000000117
Kahan summation (double)	8.0000000117

Table 1: Summation Results

Task 3. *Minimize the error*

***Solution*.** (a)
The total error ϵ_{total} is the sum of the approximation error and the machine error:

$$\epsilon_{\text{total}} = \epsilon_{\text{appr}} + \epsilon_m$$

Given $\epsilon_{\text{appr}} \approx \frac{1}{N}$ and $\epsilon_m \approx 10^{-7}$:

$$\epsilon_{\text{total}} = \frac{1}{N} + 10^{-7}$$

Differentiating:

$$\frac{d}{dN}\left(\frac{1}{N} + 10^{-7}\right) = -\frac{1}{N^2}$$

This means the larger the N, the smaller the error. However, the approximation error has upper bound of machine error.
Therefore, the minimum of the total error will be $\epsilon = 2 \times 1^{-7}$ when $N = 10^7$.

(b)
With the same approach as above,

$$N = \frac{1}{10^{-15}} = 10^{15}$$

Total error:

$$\epsilon_{\text{total}} = \frac{1}{10^{15}} + 10^{-15} = 2 \times 10^{-15}$$

□

Task 4. *Calculating a power series*

***Solution*.** The first approach is problematic for several reasons:

- Floating-point arithmetic can lose precision with very large or small numbers.
- Catastrophic cancellation can occur when summing terms of alternating signs.
- Calculating $(-x)^n$ and $n!$ from scratch in each iteration requires $O(n^2)$ operations for each term, which means it's not an efficient algorithm.

From the results, it can be seen that the good algorithm converges for negative x's, but only the small ones get to the correct answer.

Good Algorithm

x = 0.1

N	Sum(N)	Relative Error
0	1.0000000000	0.1051709181
7	0.9048374181	0.0000000000

x = 0.1000000000
Calculated: 0.9048374181
Exact: 0.9048374180
Relative Error: 0.0000000000

x = 1.0000000000

N	Sum(N)	Relative Error
0	1.0000000000	1.7182818285
13	0.3678794413	0.0000000004

x = 1.0000000000
Calculated: 0.3678794413
Exact: 0.3678794412
Relative Error: 0.0000000004

x = 10.0000000000

N	Sum(N)	Relative Error
0	1.0000000000	22025.4657948067
48	0.0000453999	0.0000000035

x = 10.0000000000
Calculated: 0.0000453999
Exact: 0.0000453999
Relative Error: 0.0000000035

x = 100.0000000000

N	Sum(N)	Relative Error
0	1.0000000000	2.6881e40
224	8.1447e25	2.1893e67

x = 100.0000000000
Calculated: 8.1447e25
Exact: 0.0000000000
Relative Error: 2.1893e67

x = 1000.0000000000

N	Sum(N)	Relative Error
0	1.0000000000	inf
348	-inf	inf

x = 1000.0000000000
Calculated: -inf

Exact: 0.0000000000
Relative Error: inf

Checking Convergence for Small Negative x

x = -0.0100000000

x	Calculated	Relative Error
-0.0100000000	1.0100501671	0.0000000000

x = -0.1000000000

x	Calculated	Relative Error
-0.1000000000	1.1051709181	0.0000000000

x = -1.0000000000

x	Calculated	Relative Error
-1.0000000000	2.7182818262	0.0000000008

Checking Convergence for Increasing |x|

x = -10.0000000000

N	Sum(N)	Relative Error
0	1.0000000000	0.9999546001
34	22026.4657475864	0.0000000021

x = -10.0000000000
Calculated: 22026.4657475864
Exact: 22026.4657948067
Relative Error: 0.0000000021

x = -100.0000000000

N	Sum(N)	Relative Error
0	1.0000000000	1.0000000000
161	2.6881e40	0.0000000126

x = -100.0000000000
Calculated: 2.6881e40
Exact: 2.6881e40
Relative Error: 0.0000000126

Bad Algorithm

x	Sum	Relative Error
-100.0000000000	nan	nan
-10.0000000000	22026.4656324230	0.0000000074
-1.0000000000	2.7182818011	0.0000000100
-0.1000000000	1.1051709167	0.0000000013
-0.0100000000	1.0100501667	0.0000000004
0.1000000000	0.9048374167	0.0000000015
1.0000000000	0.3678794392	0.0000000053
10.0000000000	0.0000453999	0.0000000002
100.0000000000	nan	nan
1000.0000000000	nan	nan

□

Task 5. *Graphical output in an X window with Xgraphics and a simple makefile*

***Solution*.** □