

Lab 06

Nạp chồng toán tử & khuôn mẫu hàm

Lập trình hướng đối tượng

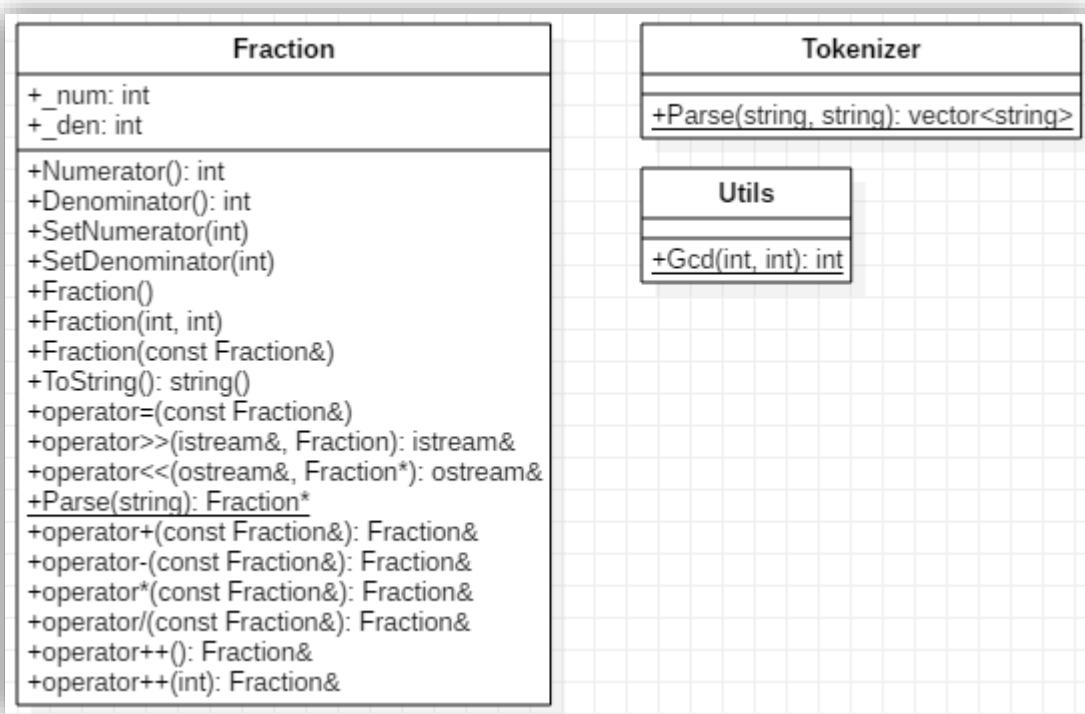
Mục tiêu

Nạp chồng các toán tử đơn giản
Làm quen với việc cài đặt khuôn mẫu hàm đơn giản

1 Hướng dẫn khởi đầu

Mô tả bài tập

Cho trước lớp phân số như thiết kế sau:



Hãy cài đặt các toán tử:

- +, -, *, /: hỗ trợ thao tác với hai phân số
- ++, --: tác động lên phân số hiện tại, cộng với 1

Hướng dẫn cài đặt

Bước 1: Cài đặt lớp hỗ trợ việc tính ước chung lớn nhất như sau

```

class Utils {
public:
    // Tìm ước số chung lớn nhất
    static unsigned Gcd(unsigned a, unsigned b) {
        if (b != 0) {
            return Gcd(b, a % b);
        }
        else {
            return a;
        }
    }
};

```

Bước 2: Cải tiến trở thành lớp Tokenizer như sau

```
class Tokenizer {  
public:  
    static vector<string> Parse(string line, string separator) {  
        vector<string> tokens;  
  
        int startPos = 0; // Bắt đầu tìm ở đầu chuỗi  
        size_t foundPos = line.find_first_of(separator, startPos);  
  
        while (foundPos != string::npos) {  
            // Tìm thấy thì cắt chuỗi con ra  
            int count = foundPos - startPos; // Số lượng kí tự  
            string token = line.substr(startPos, count);  
            tokens.push_back(token);  
  
            // Cập nhật vị trí bắt đầu tìm chuỗi mới  
            startPos = foundPos + separator.length();  
  
            // Tiếp tục tìm kiếm  
            foundPos = line.find_first_of(separator, startPos);  
        }  
  
        // Phần còn lại  
        int count = line.length() - startPos;  
        string token = line.substr(startPos, count);  
        tokens.push_back(token);  
  
        return tokens;  
    }  
};
```

Cài đặt lớp **Fraction** như sau:

```
class Fraction {  
private:  
    static string SEPERATOR;  
private:  
    int _num;  
    int _den;  
public:  
    int Numerator() { return _num; }  
    int Denominator() { return _den; }  
    void SetNumerator(int value) { _num = value; }  
    void SetDenominator(int value) { _den = value; }  
public:  
    Fraction() {  
        _num = 0;  
        _den = 1;  
    }  
  
    Fraction(int num, int den) {  
        _num = num;  
        _den = den;  
    }  
  
    Fraction(const Fraction& other) {  
        _num = other._num;  
        _den = other._den;  
    }  
  
    Fraction& operator=(const Fraction& other) {  
        _num = other._num;  
        _den = other._den;  
        return *this;  
    }  
}
```

```
public:  
    string ToString() const{  
        stringstream writer;  
        writer << _num << SEPERATOR << _den;  
        return writer.str();  
    }  
  
    friend istream& operator>>(istream& reader, Fraction& f) {  
        cout << "Nhập tu: ";  
        reader >> f._num;  
        cout << "Nhập mau: ";  
        reader >> f._den;  
        return reader ;  
    }  
  
    friend ostream& operator<<(ostream& writer, const Fraction* f) {  
        writer << f->ToString();  
        return writer;  
    }  
public:  
    static Fraction* Parse(string line) {  
        vector<string> tokens = Tokenizer::Parse(line, SEPERATOR);  
  
        int num = stoi(tokens[0]);  
        int den = stoi(tokens[1]);  
        return new Fraction(num, den);  
    }
```

```
public:  
    static Fraction* Parse(string line) {  
        vector<string> tokens = Tokenizer::Parse(line, SEPERATOR);  
  
        int num = stoi(tokens[0]);  
        int den = stoi(tokens[1]);  
        return new Fraction(num, den);  
    }  
public:  
    Fraction& operator+(const Fraction& other) {  
        int num = _num * other._den + _den * other._num;  
        int den = _den * other._den;  
        int gcd = Utils::Gcd(num, den);  
        Fraction result(num / gcd, den / gcd);  
        return result;  
    }  
  
    Fraction& operator++() {  
        _num = _num * 1 + _den * 1;  
  
        int gcd = Utils::Gcd(_num, _den);  
        _num /= gcd;  
        _den /= gcd;  
        return *this;  
    }  
};
```

```
string Fraction::SEPERATOR = "/";
```

```
int main()  
{
```

2

Bài tập vận dụng phần quá tải

Yêu cầu: Hoàn thiện các cài đặt hàm còn lại theo như thiết kế. Sau đó minh họa trong hàm main **TẤT CẢ** toán tử vừa hoàn thiện, lưu ý: đầu vào 2 phân số nhập từ bàn phím

Chú ý: operator++() là toán tử ++ ở phía **trước** (ví dụ ++i), còn định nghĩa chồng toán tử ++ ở phía **sau** (ví dụ i++) thì nguyên mẫu hàm là operator++(int) trong đó đối số int là đối số giả để phân biệt mà thôi.

3

Khuôn mẫu hàm**Mô tả bài tập**

Cài đặt hàm tìm số nhỏ nhất giữa hai số thuộc các loại: số nguyên, số thực, Phân số sử dụng function template

Hướng dẫn cài đặt**Bước 1: Tạo mới tập tin Compare.cpp chứa khai báo khuôn mẫu như sau**

```
template <class T>
T Min(T a, T b) {
    return a < b ? a : b;
}
```

Chú ý nếu đặt tên là min (viết thường) thì sẽ bị trùng với tên của hệ thống

Bước 2: Cài đặt hàm main để kiểm tra

```
int main()
{
    cout << Min(3, 4) << endl;
    cout << Min(3.14, 5.76) << endl;
    cout << Min('A', 'C') << endl;
}
```

4

Bài tập vận dụng phần khuôn mẫu

Yêu cầu: Cài đặt lớp mảng động **DynamicArray** theo function template hỗ trợ kiểu dữ liệu lớp bất kỳ. Các thao tác cần hỗ trợ:

- + **PushBack(T)** : Thêm một phần tử vào mảng
- + **T GetAt (int)** : Lấy một phần tử tại vị trí truyền vào, tự động xử lý chỉ số vượt mức đưa về vùng hợp lệ
- + **T MaxElement()**: Trả ra phần tử lớn nhất của mảng
- + **void Sort()**: hàm sắp xếp tăng mảng

Minh họa với các kiểu/lớp sau: **int** và **Fraction** trong hàm main TẤT CẢ phương thức. Lưu ý: dữ liệu đầu vào của các phương thức nhập từ bàn phím, trừ 2 phương thức "Sort" và "MaxElement"

-- HẾT --