Lab 09
Nạp chồng toán tử

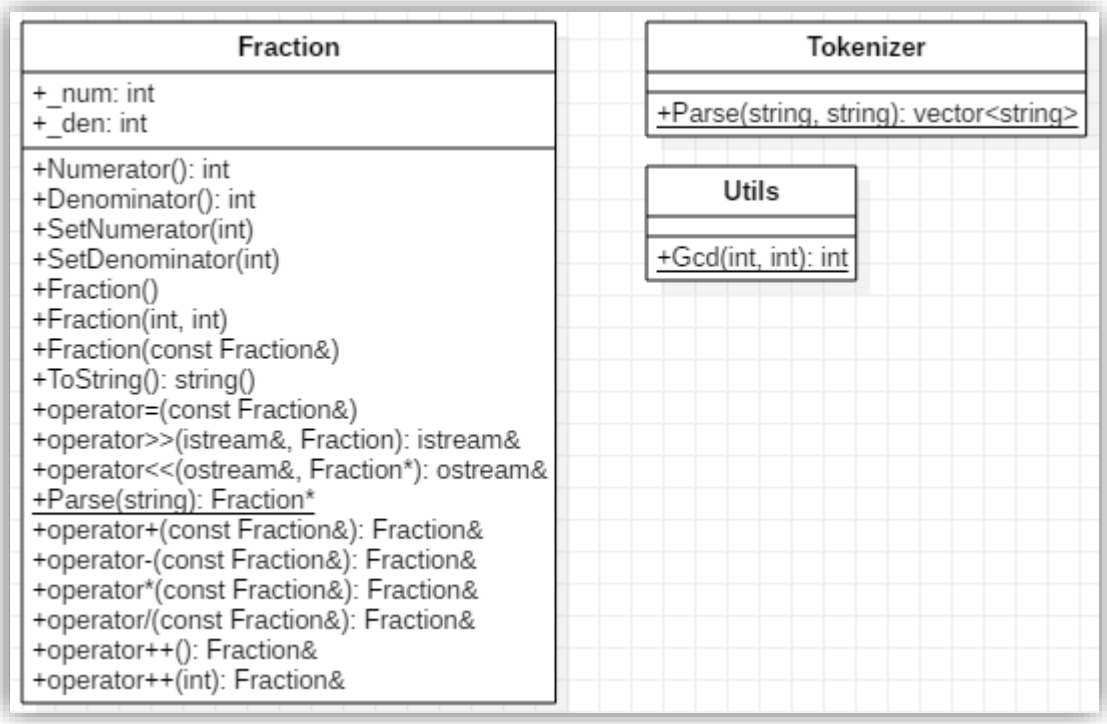
Lập trình hướng đối tượng

Mục tiêu	Nạp chồng các toán tử 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: Tạo mới dự án

- Chọn loại dự án là **C++** / **Console Application**.

- Đặt tên solution là: **OperatorOverload**. Đặt tên project là **FractionV9**

- Nếu sử dụng Visual Studio 2017 trở lên cần vô hiệu hóa **Precompiled header** bằng cách nhấn phải vào project chọn Properties. Vào mục **C / C++** > **All Options**, tìm tới tùy chọn **Precompiled header** và chọn **Not using precompiled headers**.

Bước 2: 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 3: Cải tiến trở thành lớp Tokenizer như sau

```
class Tokenizer {
public:
    static vector<string> Parse(string line, string seperator) {
        vector<string> tokens;

        int startPos = 0; // Bắt đầu tìm ở đầu chuỗi
        size_t foundPos = line.find_first_of(seperator, 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 + seperator.length();

            // Tiếp tục tìm kiếm
            foundPos = line.find_first_of(seperator, 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 << "Nhap tu:";
        reader >> f._num;
        cout << "Nhap 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

Yêu cầu

Hoàn thiện các cài đặt hàm còn lại theo như thiết kế.

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 Hướng dẫn nộp bài

Trước khi nộp cần chú ý:

- Lấy tập tin exe được biên dịch sẵn trong thư mục Debug, copy nó ra thư mục Release bên ngoài mã nguồn.

- Xóa hết tất cả các tập tin trung gian trong quá trình biên dịch bằng cách chọn **Build** > **Clean**.

- Chú ý thư mục ẩn **.vs** rất nặng. Cần hiển thị file ẩn mới thấy và xóa nó đi được.

> Nếu bạn muốn biết cách làm đúng thì cần tự tìm cách build ở chế độ Release và copy file exe kết quả ra bên ngoài để nộp mới đúng. Tuy nhiên nếu chưa hiểu ý nghĩa thì cứ lấy đại file exe có sẵn đi nộp cũng được (hiện tại đang trong thư mục Debug ứng với chế độ biên dịch Debug)

Tổ chức bài nộp

+ Thư mục **Source**: chứa mã nguồn đã được clean

+ Thư mục **Release**: chứa tập tin thực thi đã được biên dịch từ mã nguồn

+ Tập tin **readme.txt**: chứa thông tin sinh viên, gồm MSSV và họ tên. Ghi chú kèm các thông tin giáo viên cần chú ý khi chấm bài.

Để nộp bài, nén tất cả lại và đặt tên với định dạng **MSSV.zip** hoặc **MSSV.rar** và nộp.

Nếu làm đúng các bước trên file này sẽ có kích thước < 100 KB!

(Tuy nhiên cũng đừng quá lo lắng nếu nó khác con số trên, miễn < 12 MB để nộp được trên moodle là okie nhé!)

Ngoài lề: Để đảm bảo sau này nhìn vào file nén còn biết ngay nó làm gì, ta nên thêm vào một số thông tin theo sau MSSV. Ví dụ: 0712221-Lab09-FunctionOverload.zip. Tuy nhiên việc này là KHÔNG bắt buộc nhé.

-- HẾT --