Homework 5 컴퓨터공학부 202211390 최준원

Q1

Source Code

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

class Fraction
{
private:
	int numerator;
	int denominator;
public:
	Fraction();
	Fraction(int numer);
	Fraction(int numer, int denom);
	Fraction(const Fraction& fract);
	~Fraction();

	// Declaration of Unary plus(not constant)
	Fraction operator+();
	// Declaration of Unary minus(not constant)
	Fraction operator-();

	//Declaration of Prefix In/Decrement Operator
	Fraction& operator++();
	Fraction& operator--();
	//Declaration of Postfix In/Decrement Operator
	const Fraction operator++(int);
	const Fraction operator--(int);
	// Declaration of inequality operator
	const bool operator!=(const Fraction& right);

	Fraction& operator+=(int n);
	Fraction& operator-=(int n);
	Fraction& operator*=(int n);
	Fraction& operator/=(int n);

	Fraction& operator+(const Fraction& right);
	Fraction& operator-(const Fraction& right);
	Fraction& operator*(const Fraction& right);
	Fraction& operator/(const Fraction& right);

	//Chosen Operator
	const bool operator>=(const Fraction& right);

	int getNumerator() const { return numerator; }
	int getDenominator() const { return denominator; }
	void setNumerator(int numer);
	void setDenominator(int denom);
	string print();
private:
	bool normalize();
	int gcd(int n, int m);
```

```
};



int main()
{
	// Creation of two objects and testing the plus and minus operator
	Fraction fract1(2, 3);
	Fraction fract2(1, 2);
	cout << "fract1: " << fract1.print() << endl;
	cout << "fract2: " << fract2.print() << endl;
	+fract1;
	-fract2;
	cout << "Result of +fract1: " << fract1.print() << endl;
	cout << "Result of -fract2: " << fract2.print() << endl << endl;
	// Creation of four objects and testing the ++ and -- operators
	Fraction fract3(3, 4);
	Fraction fract4(4, 5);
	Fraction fract5(5, 6);
	Fraction fract6(6, 7);
	cout << "fract3: " << fract3.print() << endl;
	cout << "fract4: " << fract4.print() << endl;
	cout << "fract5: " << fract5.print() << endl;
	cout << "fract6: " << fract6.print() << endl << endl;
	++fract3;
	--fract4;
	Fraction fract55 = fract5++;
	Fraction fract66 = fract6--;
	cout << "Result of ++fract3: " << fract3.print() << endl;
	cout << "Result of --fract4: " << fract4.print() << endl;
	cout << "Result of fract5++: " << fract5.print() << endl;
	cout << "Result of fract6--: " << fract6.print() << endl << endl;
	// Testing assignment & inequality operators
	if (fract3 != fract4)
	{
		fract3 = fract4;
	}
	cout << "Result of fract3 != fract4: "
		<< to_string(fract3 != fract4) << endl;
	cout << "fract3: " << fract3.print() << endl << endl;
	// Testing compound assignment operators
	Fraction fract7(3, 5);
	Fraction fract8(4, 7);
	Fraction fract9(5, 8);
	Fraction fract10(7, 9);
	fract7 += 2; // == Fraction(2, 1)
	fract8 -= 3; // == Fraction(3, 1)
	fract9 *= 4; // == Fraction(4, 1)
	fract10 /= 5; // == Fraction(5, 1)
	cout << "Result of fract7 += 2: " << fract7.print() << endl;
	cout << "Result of fract8 -= 3: " << fract8.print() << endl;
	cout << "Result of fract9 *= 4: " << fract9.print() << endl;
	cout << "Result of fract10 /= 5: " << fract10.print() << endl << endl;
	// Testing binary arithmetic operators
	// Testing binary arithmetic operators
	Fraction fract11(3, 5);
	Fraction fract111 = fract11 + Fraction(2);
	Fraction fract112 = fract11 - Fraction(3);
	Fraction fract113 = fract11 * Fraction(4);
	Fraction fract114 = fract11 / Fraction(5);
	cout << "Result of fract11 + 2: " << fract111.print() << endl;
	cout << "Result of fract11 - 3: " << fract112.print() << endl;
	cout << "Result of fract11 * 4: " << fract113.print() << endl;
	cout << "Result of fract11 / 5: " << fract114.print() << endl << endl;
	// Q2. Demonstrate that your chosen operator works correctly by
following these steps:
	// print out “your chosen operator: “.
	// print out the input values.
	// print out the expected output from the manual.
	// print out the output from the operator overloading.
	// print out the comparison result indicating whether they are the same
or not.
	// repeat this test at least two more times.

	cout << "Chosen Operator: >=" << endl;

	//양수 대 양수
	cout << "\nTest1\nInput Values: 4/3 >= 2/5" << endl;
	Fraction fract20(4, 3);
	Fraction fract21(2, 5);
	cout << "Expected Output: 1" << endl;
	cout << "My Output: " << to_string(fract20 >= fract21) << endl;
	cout << "Answer: " << endl;
	if (to_string(fract20 >= fract21) == "1") {
		cout << "Correct" << endl;
	}
	else {
		cout << "Wrong" << endl;
	}
	//음수 대 음수
	cout << "\nTest2\nInput Values: -123/23 >= -192/129" << endl;
	Fraction fract22(-123, 23);
	Fraction fract23(-192, 129);
	cout << "Expected Output: 0" << endl;
	cout << "My Output: " << to_string(fract22 >= fract23) << endl;
	cout << "Answer: " << endl;
	if (to_string(fract22 >= fract23) == "0") {
		cout << "Correct" << endl;
	}
	else {
		cout << "Wrong" << endl;
	}
	//양수 대 음수
	cout << "\nTest3\nInput Values: 7/23 >= -12/139" << endl;
	Fraction fract24(7, 23);
	Fraction fract25(-12, 139);
	cout << "Expected Output: 1" << endl;
	cout << "My Output: " << to_string(fract24 >= fract25) << endl;
	cout << "Answer: " << endl;
	if (to_string(fract24 >= fract25) == "1") {
		cout << "Correct" << endl;
	}
	else {
		cout << "Wrong" << endl;
	}
	//음수 대 양수
	cout << "\nTest4\nInput Values: -17/23 >= 2" << endl;
	Fraction fract26(-17, 23);
	Fraction fract27(2);
	cout << "Expected Output: 0" << endl;
```

```cpp
	cout << "My Output: " << to_string(fract26 >= fract27) << endl;
	cout << "Answer: " << endl;
	if (to_string(fract26 >= fract27) == "0") {
		cout << "Correct" << endl;
	}
	else {
		cout << "Wrong" << endl;
	}
	//0이 좌변
	cout << "\nTest5\nInput Values: 0 >= 19/34" << endl;
	Fraction fract28;
	Fraction fract29(19, 34);
	cout << "Expected Output: 0" << endl;
	cout << "My Output: " << to_string(fract28 >= fract29) << endl;
	cout << "Answer: " << endl;
	if (to_string(fract28 >= fract29) == "0") {
		cout << "Correct" << endl;
	}
	else {
		cout << "Wrong" << endl;
	}
	//0이 우변
	cout << "\nTest6\nInput Values: 127/23 >= 0" << endl;
	Fraction fract30(127, 23);
	Fraction fract31;
	cout << "Expected Output: 1" << endl;
	cout << "My Output: " << to_string(fract30 >= fract31) << endl;
	cout << "Answer: " << endl;
	if (to_string(fract30 >= fract31) == "1") {
		cout << "Correct" << endl;
	}
	else {
		cout << "Wrong" << endl;
	}



	return 0;
}

Fraction::Fraction()
	: numerator(0), denominator(1)
{
}

Fraction::Fraction(int numor)
	: numerator(numor), denominator(1)
{
}

Fraction::Fraction(int numor, int denom = 1)
	: numerator(numor), denominator(denom)
{
	normalize();
}

Fraction::Fraction(const Fraction& fract)
	: numerator(fract.numerator), denominator(fract.denominator)
{
}
```

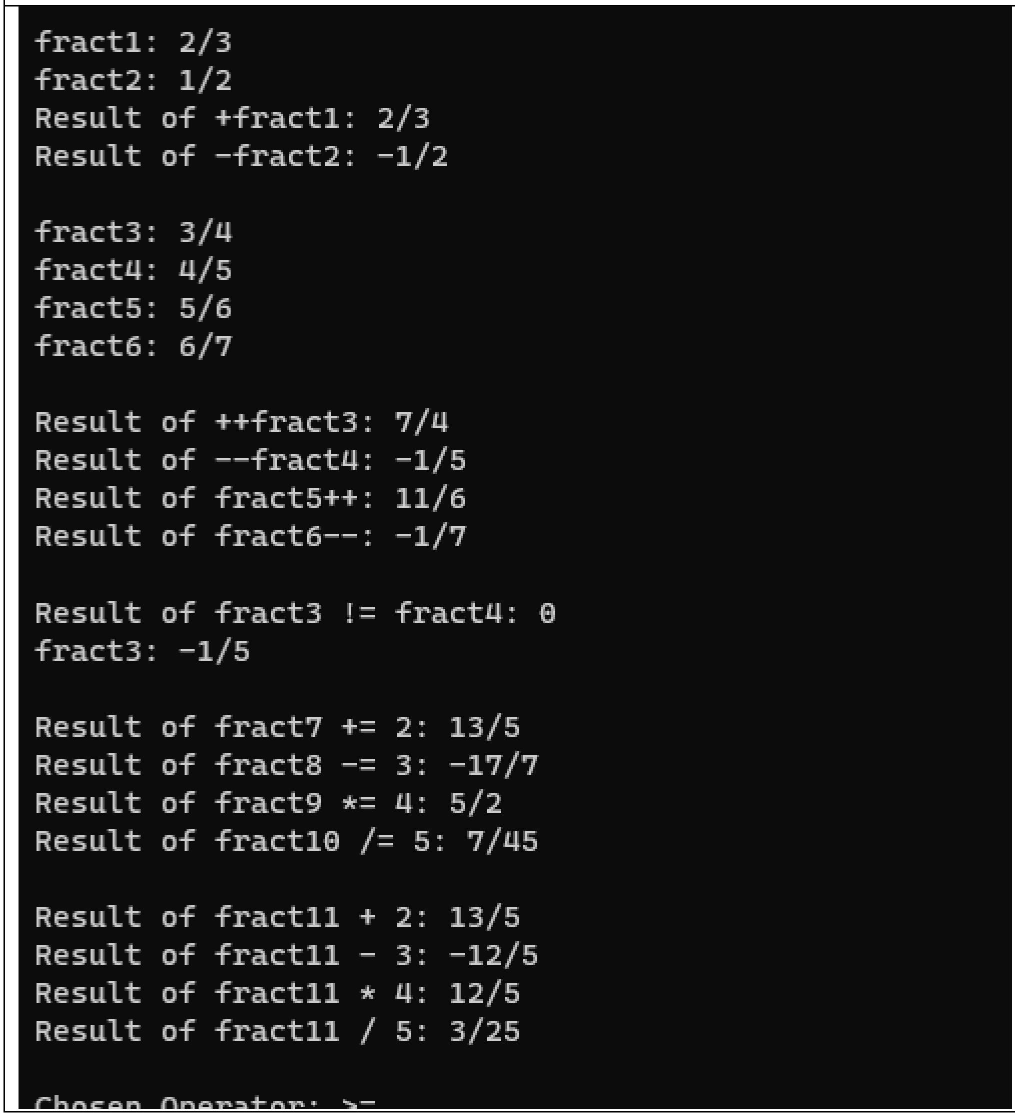
```
Fraction :: ~Fraction()
{
}
string Fraction::print()
{
       return to_string(numerator) + "/" + to_string(denominator);
}
void Fraction::setNumerator(int numer)
{
       numerator = numer;
       normalize();
}
void Fraction::setDenominator(int denom)
{
       denominator = denom;
       normalize();
}

bool Fraction::normalize()
{
       // Handling a denominator of zero
       if (denominator == 0)
       {
              cout << "Invalid denomination. Need to quit." << endl;
              return false;
       }
       // Changing the sign of denominator
       if (denominator < 0)
       {
              denominator = -denominator;
              numerator = -numerator;
       }
       // Dividing numerator and denominator by gcd
       int divisor = gcd(abs(numerator), abs(denominator));
       numerator = numerator / divisor;
       denominator = denominator / divisor;
       return true;
}
int Fraction::gcd(int n, int m)
{
       int gcd = 1;
       for (int k = 1; k <= n && k <= m; k++)
       {
              if (n % k == 0 && m % k == 0)
              {
                     gcd = k;
              }
       }
       return gcd;
}

// Definition of Unary plus operator
Fraction Fraction :: operator+ ()
{
       //Fraction temp(+numerator, denominator); // a new object
       //return temp;
       if (numerator >= 0) {
              this->normalize();
              return *this;
       }
       else {
```

```
                numerator = -numerator;
                this->normalize();
                return *this;
        }
}

// Definition of Unary minus operator
Fraction Fraction :: operator- ()
{
        //Fraction temp(-numerator, denominator); // a new object
        //return temp;
        numerator = -numerator;
        this->normalize();
        return *this;
}

// Definition pre-increment operator
Fraction& Fraction :: operator++()
{
        numerator = numerator + denominator;
        this->normalize();
        return *this;
}

// Definition pre-decrement operator
Fraction& Fraction :: operator--()
{
        numerator = numerator - denominator;
        this->normalize();
        return *this;
}

// Definition of post-increment operator
const Fraction Fraction :: operator++(int dummy)
{
        Fraction temp(numerator, denominator);
        ++(*this);
        return temp;
}

// Definition of post-decrement operator
const Fraction Fraction :: operator--(int dummy)
{
        Fraction temp(numerator, denominator);
        --(*this);
        return temp;
}

// Definition of inequality operator
const bool Fraction :: operator!=(const Fraction & right)
{
        return this->numerator * right.denominator != right.numerator * this-
>denominator;
}

// Definition of += operator
Fraction& Fraction :: operator+=(int n)
{
        Fraction right(n, 1);
        numerator = numerator * right.denominator + denominator *
right.numerator;
```

```
	denominator = denominator * right.denominator;
	normalize();
	return *this;
}
// Definition of -= operator
Fraction& Fraction :: operator-=(int n)
{
	Fraction right(n, 1);
	numerator = numerator * right.denominator - denominator *
right.numerator;
	denominator = denominator * right.denominator;
	normalize();
	return *this;
}
// Definition of *= operator
Fraction& Fraction :: operator*=(int n)
{
	Fraction right(n, 1);
	numerator = numerator * right.numerator;
	denominator = denominator * right.denominator;
	normalize();
	return *this;
}
// Definition of /= operator
Fraction& Fraction :: operator/=(int n)
{
	Fraction right(n, 1);
	//numerator = numerator * right.denominator / denominator *
right.numerator;
	denominator = denominator * right.numerator;
	normalize();
	return *this;
}

// Definition of + operator
Fraction& Fraction :: operator+(const Fraction& right)
{
	//Fraction right(n, 1);
	Fraction temp(numerator, denominator);
	temp.numerator = temp.numerator + temp.denominator * right.numerator;
	temp.denominator = temp.denominator * right.denominator;
	normalize();
	return temp;
}
// Definition of - operator
Fraction& Fraction :: operator-(const Fraction& right)
{
	Fraction temp(numerator, denominator);
	temp.numerator = temp.numerator - temp.denominator * right.numerator;
	temp.denominator = temp.denominator * right.denominator;
	normalize();
	return temp;
}
// Definition of * operator
Fraction& Fraction :: operator*(const Fraction& right)
{
	Fraction temp(numerator, denominator);
	temp.numerator = temp.numerator * right.numerator;
	temp.denominator = temp.denominator * right.denominator;
	normalize();
	return temp;
```

```
}
// Definition of / operator
Fraction& Fraction :: operator/(const Fraction& right)
{
       Fraction temp(numerator, denominator);
       temp.numerator = temp.numerator * right.denominator;
       temp.denominator = temp.denominator * right.numerator;
       normalize();
       return temp;
}
// Definition of >= operator
const bool Fraction :: operator>=(const Fraction& right) {
       return this->numerator* right.denominator >= right.numerator * this-
>denominator;
}
```

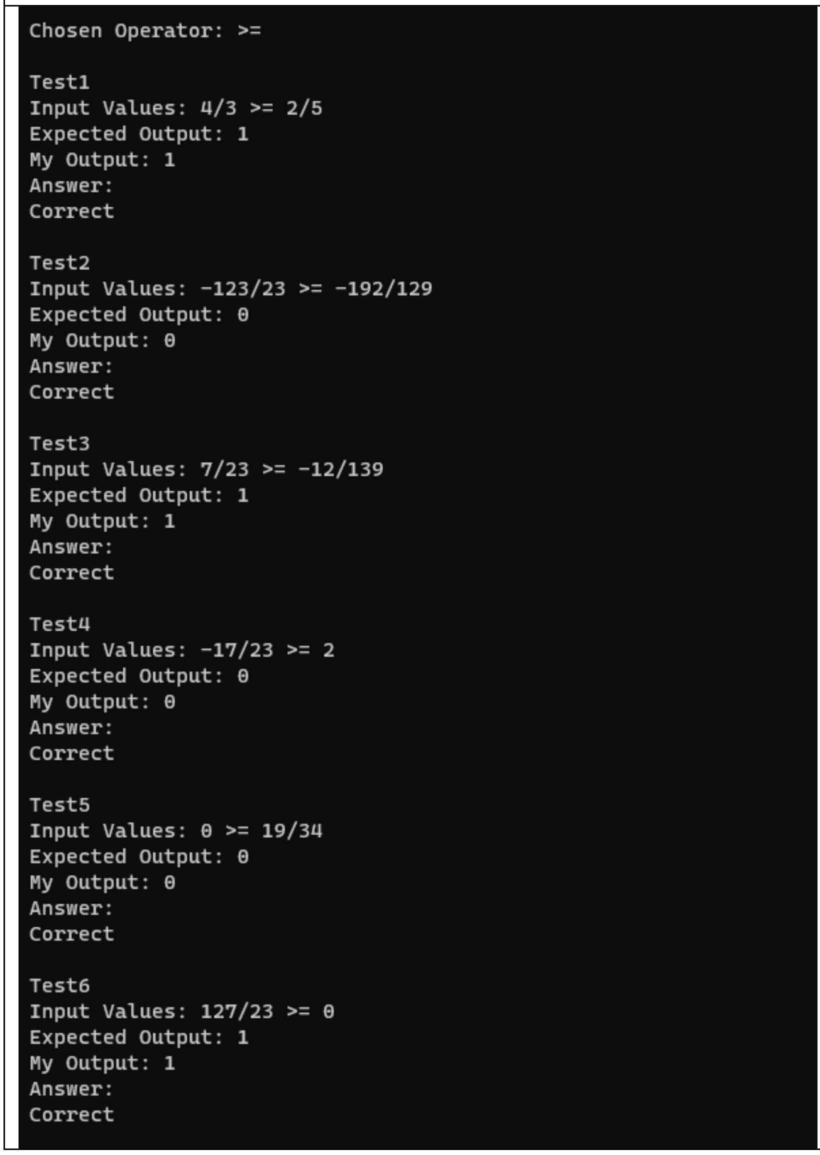
Screenshot



```
fract1: 2/3
fract2: 1/2
Result of +fract1: 2/3
Result of -fract2: -1/2

fract3: 3/4
fract4: 4/5
fract5: 5/6
fract6: 6/7

Result of ++fract3: 7/4
Result of --fract4: -1/5
Result of fract5++: 11/6
Result of fract6--: -1/7

Result of fract3 != fract4: 0
fract3: -1/5

Result of fract7 += 2: 13/5
Result of fract8 -= 3: -17/7
Result of fract9 *= 4: 5/2
Result of fract10 /= 5: 7/45

Result of fract11 + 2: 13/5
Result of fract11 - 3: -12/5
Result of fract11 * 4: 12/5
Result of fract11 / 5: 3/25

Chosen Operator: >=
```

Q2

Screenshot



```
Chosen Operator: >=

Test1
Input Values: 4/3 >= 2/5
Expected Output: 1
My Output: 1
Answer:
Correct

Test2
Input Values: -123/23 >= -192/129
Expected Output: 0
My Output: 0
Answer:
Correct

Test3
Input Values: 7/23 >= -12/139
Expected Output: 1
My Output: 1
Answer:
Correct

Test4
Input Values: -17/23 >= 2
Expected Output: 0
My Output: 0
Answer:
Correct

Test5
Input Values: 0 >= 19/34
Expected Output: 0
My Output: 0
Answer:
Correct

Test6
Input Values: 127/23 >= 0
Expected Output: 1
My Output: 1
Answer:
Correct
```