**202211341 컴퓨터공학부 이윤희**

|  |
| --- |
| **In-class Exercise** |
| **Source code** |
| #include <iostream>  #include <string>  using namespace std;  class Fraction  {  private:  int numerator;  int denominator;  public:  Fraction();  Fraction(int n);  Fraction(int numer, int denom);  Fraction(const Fraction& fract);  ~Fraction();  int getNumerator() const { return numerator; }  int getDenominator() const { return denominator; }  void setNumerator(int numer);  void setDenominator(int denom);  string print() const;  Fraction& operator+();  Fraction& operator-();  Fraction& operator++();  Fraction& operator--();  const Fraction operator++(int);  const Fraction operator--(int);  const bool operator!=(const Fraction& right);  Fraction& operator=(const Fraction& right);  Fraction& operator+=(const int right);  Fraction& operator-=(const int right);  Fraction& operator\*=(const int right);  Fraction& operator/=(const int right);  Fraction& operator+(const Fraction& right) const;  Fraction& operator-(const Fraction& right) const;  Fraction& operator\*(const Fraction& right) const;  Fraction& operator/(const Fraction& right) const;  bool operator<(const Fraction& right) const;  private:  bool normalize();  int gcd(int n, int m);  };  Fraction::Fraction()  : numerator(0), denominator(1) {  }  Fraction::Fraction(int n)  :numerator(n), denominator(1) {  }  Fraction::Fraction(int numor, int denom)  : numerator(numor), denominator(denom) {  normalize();  }  Fraction::Fraction(const Fraction& fract)  : numerator(fract.numerator), denominator(fract.denominator) {  }  Fraction :: ~Fraction() {  }  string Fraction::print() const {  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;  }  Fraction& Fraction::operator+()  {  numerator = +numerator;  this->normalize();  return \*this;  }  Fraction& Fraction::operator-()  {  numerator = -numerator;  this->normalize();  return \*this;  }  Fraction& Fraction::operator++() {  numerator = numerator + denominator;  this->normalize();  return \*this;  }  Fraction& Fraction::operator--() {  numerator = numerator - denominator;  this->normalize();  return \*this;  }  const Fraction Fraction::operator++(int) {  Fraction temp(numerator, denominator);  ++(\*this);  return temp;  }  const Fraction Fraction::operator--(int) {  Fraction temp(numerator, denominator);  --(\*this);  return temp;  }  const bool Fraction::operator!=(const Fraction& right) {  return this->numerator \* right.denominator != right.numerator \* this->denominator;  }  Fraction& Fraction::operator=(const Fraction& right)  {  if (\*this != right) //or check in another way  {  numerator = right.numerator;  denominator = right.denominator;  }  return \*this;  }  Fraction& Fraction :: operator+=(const int right)  {  Fraction r = Fraction(right, 1);  numerator = numerator \* r.denominator + denominator \* r.numerator;  denominator = denominator \* r.denominator;  normalize();  return \*this;  }  Fraction& Fraction :: operator-=(const int right)  {  Fraction r = Fraction(right, 1);  numerator = numerator \* r.denominator - denominator \* r.numerator;  denominator = denominator \* r.denominator;  normalize();  return \*this;  }  Fraction& Fraction :: operator\*=(const int right)  {  Fraction r = Fraction(right, 1);  numerator = numerator \* r.numerator;  denominator = denominator \* r.denominator;  normalize();  return \*this;  }  Fraction& Fraction :: operator/=(const int right)  {  Fraction r = Fraction(right, 1);  numerator = numerator \* r.denominator;  denominator = denominator \* r.numerator;  normalize();  return \*this;  }  Fraction& Fraction::operator+(const Fraction& right) const  {  int num = this->numerator \* right.denominator + right.numerator \* this->denominator;  int denom = this->denominator \* right.denominator;  Fraction temp(num, denom);  return temp;  }  Fraction& Fraction::operator-(const Fraction& right) const  {  int num = this->numerator \* right.denominator - right.numerator \* this->denominator;  int denom = this->denominator \* right.denominator;  Fraction temp(num, denom);  return temp;  }  Fraction& Fraction::operator\*(const Fraction& right) const  {  int num = this->numerator \* right.numerator;  int denom = this->denominator \* right.denominator;  Fraction temp(num, denom);  return temp;  }  Fraction& Fraction::operator/(const Fraction& right) const  {  int num = this->numerator \* right.denominator;  int denom = this->denominator \* right.numerator;  Fraction temp(num, denom);  return temp;  }  bool Fraction::operator<(const Fraction& right) const {  if (numerator \* right.denominator < denominator \* right.numerator)  return true;  else  return false;  }  int main(void) {  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;  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;  if (fract3 != fract4) {  fract3 = fract4;  }  cout << "Result of fract3 != fract4: "  << to\_string(fract3 != fract4) << endl;  cout << "fract3: " << fract3.print() << endl << endl;  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;  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;  Fraction f1(1, 2); Fraction f2(1, 3);  cout << "my chosen operator: " << "<" << endl;  cout << "input: " << f1.print() << ", " << f2.print() << endl;  cout << "expected output: " << false << endl;  cout << "output: " << (f1 < f2) << endl;  cout << "the results: ";  if (f1 < f2 == false)  cout << "same" << endl << endl;  else  cout << "different" << endl << endl;  Fraction f3(78, 99); Fraction f4(76, 101);  cout << "my chosen operator: " << "<" << endl;  cout << "input: " << f3.print() << ", " << f4.print() << endl;  cout << "expected output: " << false << endl;  cout << "output: " << (f3 < f4) << endl;  cout << "the results: ";  if (f3 < f4 == false)  cout << "same" << endl << endl;  else  cout << "different" << endl << endl;  Fraction f5(341, 57); Fraction f6(345, 56);  cout << "my chosen operator: " << "<" << endl;  cout << "input: " << f5.print() << ", " << f6.print() << endl;  cout << "expected output: " << true << endl;  cout << "output: " << (f5 < f6) << endl;  cout << "the results: ";  if (f5 < f6 == true)  cout << "same" << endl << endl;  else  cout << "different" << endl << endl;  Fraction f7(-1, 2); Fraction f8(0, 2);  cout << "my chosen operator: " << "<" << endl;  cout << "input: " << f7.print() << ", " << f8.print() << endl;  cout << "expected output: " << true << endl;  cout << "output: " << (f7 < f8) << endl;  cout << "the results: ";  if (f7 < f8 == true)  cout << "same" << endl << endl;  else  cout << "different" << endl << endl;  return 0;  } |
| **Output** |