**Practica de Laboratorio de Objetos y Abstracción de Datos**

**Ciencia de la Computación - UNSA**

1. Considerar la clase Fraction y su implementación:

fraction.h

|  |
| --- |
| #include<iostream>  using namespace std;  class Fraction {  public:  Fraction();  Fraction(int num,int denom);  void add(Fraction f);  void mult(Fraction f);  float decimal();  int getNum();  int getDenom();  friend ostream& operator<<(ostream& out, Fraction &frac);  private:  int num; // the numerator  int denom; // the denominator  void reduce(); // reduce the fraction to lowest terms  int gcd(int u, int v);  }; |

fraction.cpp

|  |
| --- |
| #include <iostream>  #include "fraction.h"  Fraction::Fraction()  {  num = 1;  denom = 1;  }  // purpose: an overloaded constructor  Fraction::Fraction(int n, int d)  {  num = n;  denom = d;  // reduce in case we were given an unreduced fraction  reduce();  }  // getters for numerator and denominator  int Fraction::getNum() {  return num;  }  int Fraction::getDenom() {  return denom;  }  // purpose: to add another fraction  void Fraction::add(Fraction other) {  // calculate the new numerator  int new\_num = num \* other.denom + other.num \* denom;  // calculate the new denominator  int new\_denom = denom \* other.denom;  // replace our values with the new values  num = new\_num;  denom = new\_denom;  // reduce the fraction  reduce();  }  // purpose: to multiply another fraction  // with the result being stored in this fraction  void Fraction::mult(Fraction other)  {  num \*= other.num;  denom \*= other.denom;  // reduce the fraction  reduce();  }  // purpose: To return a decimal value of this fraction  float Fraction::decimal()  {  return (float)num / denom;  }  // purpose: To reduce the fraction  void Fraction::reduce() {  // reduce the fraction to lowest terms find the greatest common divisor  int frac\_gcd = gcd(num,denom);  // reduce by dividing num and denom by the gcd  num = num / frac\_gcd;  denom = denom / frac\_gcd;  }  // purpose: To recursively find the greatest common divisor of the two integers  int Fraction::gcd(int u, int v) {  if (v != 0) {  return gcd(v,u%v);  }  else {  return u;  }  }  // purpose: To overload the << operator  ostream& operator<<(ostream& out, Fraction &frac) {  out << frac.num << "/" << frac.denom;  return out;  } |

1. Escriba una función pública reciprocal que se agregará a la clase Fraction que convierta la función a su recíproca (notar que por definición el recíproco de un número x es un número y tal que xy = 1).

**void Fraction::reciprocal() { . . .**

1. Escriba una función pública divide que se agregará a la clase Fraction que divide la Fracción por una Fracción dada.

**void Fraction::divide(Fraction f) { . . .**

1. Adicionar los operadores relacionales ==, !=, <, <=, >, >=
2. Reescriba la implementación del algoritmo merge sort de abajo para que ordene un array en lugar de un vector. Su función debe usar el prototipo

void sort(int \*array, int n)

|  |
| --- |
| /\*  \* The merge sort algorithm consists of the following steps:  \*  \* 1. Divide the vector into two halves.  \* 2. Sort each of these smaller vectors recursively.  \* 3. Merge the two vectors back into the original one.  \*/  void sort(vector<int> & vec) {  int n = vec.size();  if (n <= 1) return;  vector<int> v1;  vector<int> v2;  for (int i = 0; i < n; i++) {  if (i < n / 2) {  v1.pushback(vec[i]);  } else {  v2. pushback(vec[i]);  }  }  sort(v1);  sort(v2);  vec.clear();  merge(vec, v1, v2);  } |

|  |
| --- |
| /\*  \* Function: merge  \* ---------------  \* This function merges two sorted vectors (v1 and v2) into the  \* vector vec, which should be empty before this operation.  \* Because the input vectors are sorted, the implementation can  \* always select the first unused element in one of the input  \* vectors to fill the next position.  \*/  void merge(vector<int> & vec, vector<int> & v1, vector<int> & v2) {  int n1 = v1.size();  int n2 = v2.size();  int p1 = 0;  int p2 = 0;  while (p1 < n1 && p2 < n2) {  if (v1[p1] < v2[p2]) {  vec. pushback(v1[p1++]);  } else {  vec. pushback(v2[p2++]);  }  }  while (p1 < n1) vec. pushback(v1[p1++]);  while (p2 < n2) vec. pushback(v2[p2++]);  } |