|  |
| --- |
| **BI1000. The Rational class (version 1)** |
| |  |  |  |  | | --- | --- | --- | --- | | Total: | 808 | Accepted: | 105 | |
|  |
|  |
| **Time Limit: 1sec    Memory Limit:256MB**  **Description**  Implement the Rational class using the following interface: class Rational { public:   Rational();   Rational(long numerator, long denominator);   long getNumerator();   long getDenominator();   Rational add(Rational &secondRational);   Rational subtract(Rational &secondRational);   Rational multiply(Rational &secondRational);   Rational divide(Rational &secondRational);   /\* Returns an int value -1, 0, or 1 to indicate whether this rational      number is less than, equal to, or greater than the specified number.   \*/   int compareTo(Rational &secondRational);   bool equals(Rational &secondRational);   int intValue(); //Returns the numerator/denominator   double doubleValue(); //Returns 1.0\*numberator/denominator   string toString();  private:   long numerator; //分子.    long denominator; //分母，不能为0.    static long gcd(long n, long d); }; Rational::Rational() {   numerator = 0;   denominator = 1; } string Rational::toString() {   char s[50];   if ( denominator == 1) {       sprintf(s,"%ld",numerator);   } else {     sprintf(s,"%ld/%ld",numerator,denominator);   }   return string(s); } |

|  |
| --- |
| **BI1001. The Rational class (version 2)** |
| |  |  |  |  | | --- | --- | --- | --- | | Total: | 353 | Accepted: | 61 | |
|  |
|  |
| **Time Limit: 1sec    Memory Limit:256MB**  **Description**  Implement the Rational class using the following interface: class Rational { public:   Rational();   Rational(long numerator, long denominator);   string toString();    // Define function operators for relational operators   bool operator<(const Rational &secondRational) const;   bool operator<=(const Rational &secondRational) const;   bool operator>=(const Rational &secondRational) const;   bool operator!=(const Rational &secondRational) const;   bool operator==(const Rational &secondRational) const;    // Define function operators for arithmetic operators   Rational operator+(const Rational &secondRational) const;   Rational operator-(const Rational &secondRational) const;   Rational operator\*(const Rational &secondRational) const;   Rational operator/(const Rational &secondRational) const;    // Define function operators for shorthand operators   Rational operator+=(const Rational &secondRational);   Rational operator-=(const Rational &secondRational);   Rational operator\*=(const Rational &secondRational);   Rational operator/=(const Rational &secondRational);    // Define function operator []   long& operator[](const int &index);    // Define function operators for prefix ++ and --   Rational operator++();   Rational operator--();    // Define function operators for postfix ++ and --   Rational operator++(int dummy);   Rational operator--(int dummy);    // Define function operators for unary + and -   Rational operator+();   Rational operator-();    // Define the output and input operator   friend ostream &operator<<(ostream &stream, Rational &rational);   friend istream &operator>>(istream &stream, Rational &rational);    // Define function operator for conversion   operator double();      // Add any member needed here  private:   long numerator;   long denominator;   static long gcd(long n, long d); }; Rational::Rational() {    numerator = 0;     denominator = 1;  } string Rational::toString() {   char s[50];   if ( denominator == 1) {       sprintf(s,"%ld",numerator);   } else {     sprintf(s,"%ld/%ld",numerator,denominator);   }   return string(s); } ostream& operator<<(ostream &str, Rational &rational) {   cout << rational.toString();   return str; } |

|  |
| --- |
| **BI1002. The Complex class** |
| |  |  |  |  | | --- | --- | --- | --- | | Total: | 430 | Accepted: | 83 | |
|  |
|  |
| **Time Limit: 1sec    Memory Limit:256MB**  **Description**  Implement the Complex class: class Complex {   double re; //real part of a complex number   double im; //imaginary part of a complex number public:   double real() const { return re; }   double imag() const { return im; }      // add any other function needed here }; The Complex class implemented may be used like this: void f() {   Complex a;     cout << a << endl; //output (0,0)       Complex b = Complex(1,2.5);    cout << b << endl; //output(1,2.5)      Complex c(3.0);    cout << c << endl; //output (3,0)      c += a;    cout << c << endl;       c = c + a;    cout << c << endl;       c = c + 2.5;    cout << c << endl;       c = 2.5 + c;    cout << c << endl;     c -= a;   cout << c << endl;       c = c - a;   cout << c << endl;     c = c - 2.5;   cout << c << endl;       c = 2.5 - c;   cout << c << endl;       c \*= b;   cout << c << endl;       c = c \* b;   cout << c << endl;     c = c \* 2.5;   cout << c << endl;       c = 2.5 \* c;   cout << c << endl;       c /= b;   cout << c << endl;       c = c / b;   cout << c << endl;     c = c / 2.5;   cout << c << endl;       c = 2.5 / c;   cout << c << endl;       c = a+2.5+a + b\*2.5\*b;   cout << c << endl;    c = -b;   cout << c << endl;       cout << (a==a) << endl; //output 1   cout << (a==0.0) << endl; //output 1   cout << (0.0==a) << endl; //output 1   cout << (a!=a) << endl; //output 0    cout << (a!=0.0) << endl; //output 0   cout << (0.0!=a) << endl; //output 0      cin >> c; //输入格式(a,b)表示复数a + b i   cout << c << endl; } |

|  |
| --- |
| **BI1003. Rabbit** |
| |  |  |  |  | | --- | --- | --- | --- | | Total: | 131 | Accepted: | 42 | |
|  |
|  |
| **Time Limit: 1sec    Memory Limit:32MB**  **Description**  The rabbits have powerful reproduction ability. One pair of adult rabbits can give birth to one pair of kid rabbits every month. And after m months, the kid rabbits can become adult rabbits.  As we all know, when m=2, the sequence of the number of pairs of rabbits in each month is called Fibonacci sequence. But when m<>2, the problem seems not so simple. You job is to calculate after d months, how many pairs of the rabbits are there if there is exactly one pair of adult rabbits initially. You may assume that none of the rabbits dies in this period.  **Input**  The input may have multiple test cases. In each test case, there is one line having two integers m(1<=m<=10), d(1<=d<=100), m is the number of months after which kid rabbits can become adult rabbits, and d is the number of months after which you should calculate the number of pairs of rabbits. The input will be terminated by m=d=0.  **Output**  You must print the number of pairs of rabbits after d months, one integer per line.  **Sample Input**  http://soj.sysu.edu.cn/images/clipboard.jpgCopy sample input to clipboard  2 3  3 5  1 100  0 0  **Sample Output**  5  9  1267650600228229401496703205376 |