1. Write the number of the definition on the right next to the term it defines.

   (a) **function** 10
   (b) **parameter** 4
   (c) **object** 3
   (d) **overload** 1
   (e) **parser** 9
   (f) **variable** 6
   (g) **reference** 7
   (h) **declaration** 11
   (i) **scope** 5
   (j) **byte** 8
   (k) **type** 2

   (1) Defining two functions or operators with the same name but different argument (operand) types.
   (2) A set of possible values and a set of operations (for an object).
   (3) Some memory that holds a value of a given type.
   (4) A declaration of an explicit input to a function through which a function can access the arguments passed by name.
   (5) The region of program text (source code) in which a name can be referred to.
   (6) A named object of a given type; contains a value unless uninitialized.
   (7) A value describing the location of a typed value in memory.
   (8) The basic unit of addressing in most computers.
   (9) A program that reads a stream of tokens according to a grammar.
   (10) A named unit of code that can be invoked (called) from different parts of a program; a logical unit of computation.
   (11) The specification of a name with its type in a program.

2. Recall that the Euclidean algorithm, one of the oldest algorithms in common use, can be used to reduce fractions to their simplest form, and is based on the principle that the greatest common divisor of two numbers does not change if the larger number is replaced by its difference with the smaller number. For example, 21 is the GCD of 252 and 105 ($252 = 21 \times 12$ and $105 = 21 \times 5$), and the same number 21 is also the GCD of 105 and $147 = 252 - 105$. Since this replacement reduces the larger of the two numbers, repeating this process gives successively smaller pairs of numbers until the two numbers become equal. When that occurs, they are the GCD of the original two numbers.

   A more efficient version of the algorithm shortcuts these steps, instead replacing the larger of the two numbers by its remainder when divided by the smaller of the two (with this version, the algorithm stops when reaching a zero remainder).

   (a) What recursive function call on line 4 will make carry out Euclid's algorithm for the greatest common divisor (`gcd`) of positive integers `a` and `b`?

```
1 int gcd(int a, int b) {
      if(a<b) swap(a,b);
3     if(b==0) return a;
      return gcd(b, a%b); // recursive call
5 }
```

   (b) Complete the chain of function calls leading to the greatest common divisor of 252 and 105:
   `gcd(252,105) = gcd(105,42) = gcd(42,21) = gcd(21,0) = 21`

3. Write a program that consists of a while-loop that (each time around the loop) gets an `int` from the console and then adds its reciprocal to an accumulating sum of reciprocals. Exit the program when the user enters a terminating '0'.
Precondition: A sequence of integers: $a_0, a_1, \ldots, a_n$
Postcondition: An approximating decimal for the sum of their reciprocals: $\frac{1}{a_0} + \frac{1}{a_1} + \cdots + \frac{1}{a_n}$
For example, if the user enters `2 3 2 0` the program computes a decimal approximation and returns 1.333333.
SOLN:

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

double sumReciprocals() {
    int z{0};
    double sum{0};
    while(cin>>z && z != 0) {
        sum += (double)1/z;
    }
    return sum;
}

int main() {
    cout << "\nEnter␣integers␣terminating␣with␣a␣'0'␣"
         << "\nto␣compute␣the␣sum␣of␣their␣reciprocals:␣" << endl;
    cout << sumReciprocals();
}
```

```
Enter integers terminating with a '0'
to compute the sum of their reciprocals:
2 3 2 0
1.33333
```

4. Modify the program in question 3 above to compute the numerator and denominator of the sum of reciprocals in reduced form.
Precondition: A sequence of integers: $a_0, a_1, \ldots, a_n$
Postcondition: Numerator and denominator of the sum of the reciprocals: $\frac{1}{a_0} + \frac{1}{a_1} + \cdots + \frac{1}{a_n}$ in reduced form.
For example, if the user enters `2 3 4 4 0` the program computes returns `4/3`. You can use the `gcd()` function as defined in problem 2 without rewriting it.
SOLN:

```
int gcd(int a, int b) {
    if(a<b) swap(a,b);
    if(b==0) return a;
    return gcd(b, a%b);
}

void computeRat();
int main() {
    cout << "\nEnter␣a␣sequence␣of␣natural␣numbers␣terminating"
         << "\nwith␣'0'␣to␣compute␣the␣sum␣of␣their␣reciprocals"
         << "\nas␣a␣rational␣number␣in␣reduced␣form:" << endl;
    computeRat();
}
```

```
void computeRat() {
    int a, reduceBy;
    vector<int> vint;
    while(cin >> a && a!=0) {
        vint.push_back(a);
    }
    int num{0}, den{1};
    for(int i = 0; i < vint.size(); ++i) {
        num = vint[i]*num + den;
        den *= vint[i];
        reduceBy = gcd(den, num);
        den /= reduceBy;
        num /= reduceBy;
    }
    cout << "\nsum = " << num << "/" << den;
}
```

Produces this output:

```
Enter a sequence of natural numbers terminating
with '0' to compute the sum of their reciprocals
as a rational number in reduced form:
2 3 4 4 0

sum = 4/3
////////////////////

Enter a sequence of natural numbers terminating
with '0' to compute the sum of their reciprocals
as a rational number in reduced form:
2 3 4 5 6 7 8 9 10 0

sum = 4861/2520
```

5. Consider the following program for computing continued fractions of the form $a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}}$

```
vector<int> getInts()
{   vector<int> v;
    int a;
    cout << "\nEnter a sequence of positive integers terminated by '0': ";
    while(cin >> a && a!=0) {
        v.push_back(a);
    }
    return v;
}
int cfraction(const vector<int>& v)
{   int oldDen, num{1}, den{v[v.size()-1]};
    for(int i = v.size()-2; i >= 0; --i) {
        oldDen = den;
        den = v[i]*den+num;
        num = oldDen;
```

```
16          cout << "\ncontinued␣fraction␣=␣" << num << "/" << den;
        }  // need to swap num and den at the very end.
18      cout << "\nFinal␣result␣=␣" << den << '/' << num;
}
20  int main()
{   vector<int> vint = getInts();
22      cfraction(vint);
}
```

(a) Describe the return type of the `getInts()` function.
ANS: The function `getInts()` returns a `vector` of `int`s that was input by the user using the method on lines 5-7.

(b) Describe in detail how the `while()` loop of `getInts()` works and what it does.
The loop repeats the operation of getting valid `int`s from the console while those integers are not zero (0) and pushing those integers onto the `vector` of `int`s, `v`.

(c) Describe the parameter list of the `cfraction()` function. How does this compare with pass-by-value?
ANS: The input to the `cfraction()` function is a constant reference to a `vector` of `int`. This protects the value of the parameter in the same way that passing by value, but doesn't make a copy. An effect of this is that you can't change the parameter in `cfraction()`.

(d) Suppose the user enters `1 1 1 1 1 1 0`. Thread values of `num`, `den` and `oldDen` as the program executes.

i	num	den	oldDen
	1	1	
4	1	$1 \cdot 1 + 1 = 2$	1
3	2	$1 \cdot 2 + 1 = 3$	2
2	3	$1 \cdot 3 + 2 = 5$	3
1	5	$1 \cdot 5 + 3 = 8$	5
0	8	$1 \cdot 8 + 5 = 13$	8

(e) What is printed to the console if the user enters `1 1 1 1 1 1 0`?

```
Enter a sequence of positive integers terminated by '0': 1 1 1 1 1 1 0

continued fraction = 1/2
continued fraction = 2/3
continued fraction = 3/5
continued fraction = 5/8
continued fraction = 8/13
Final result = 13/8
```

6. The incomplete program below is designed to create and test `Triangle` class:

```
1   class Point {
    private:
3       double x;
        double y;
5   public:
        Point(double xin, double yin) : /* 6a: write initiator list */ {}
7       // 6b: provide getter functions
    };
9   ostream& operator<<(ostream& os, Point p)
    {   os << '(' << p.getx() << ',' << p.gety() << ')';
11      return os;
    }
13  double distance(Point p1, Point p2)
```

```
{    // 6c: write formula for distance between points
}
class Triangle {
private:
    vector<Point> pts{Point(0,0),Point(0,0),Point(0,0)};
    vector<double> sides{3};
public:
    Triangle(vector<Point> trip) : pts(trip) {
        // 6d: construct sides
    }
    Point getP1() { return pts[0]; }
    Point getP2() { return pts[1]; }
    Point getP3() { return pts[2]; }
    // 6e: write getter functions for side lengths
};
int main() {
    Point p1(0,0);
    Point p2(0,1);
    Point p3(1,0);
    vector<Point> vP;
    vP.push_back(p1);
    vP.push_back(p2);
    vP.push_back(p3);
    Triangle T1(vP);
    cout << T1.getP1() << endl;
    cout << T1.getP2() << endl;
    cout << T1.getP3() << endl;
    cout << "\nside␣a␣=␣" << T1.getS1();
    cout << "\nside␣b␣=␣" << T1.getS2();
    cout << "\nside␣c␣=␣" << T1.getS3();
}
```

(a) Write initiator list for the `Point` constructor.
ANS: `Point(double xin, double yin) : x(xin), y(yin) {}`

(b) Write the getter functions for the `Point` class.
ANS:

```
double getx() { return x; }
double gety() { return y; }
```

(c) Define the `distance()` function to work with the `Point` class.
ANS:

```
double distance(Point p1, Point p2) {
    return sqrt((p1.getx()-p2.getx())*(p1.getx()-p2.getx())+
                (p1.gety()-p2.gety())*(p1.gety()-p2.gety()));
}
```

(d) Complete the `Triangle` constructor to define the `sides` (use the distance function you wrote.)
ANS:

```
Triangle(vector<Point> trip) : pts(trip) {
    sides[0] = distance(pts[1],pts[2]);
    sides[1] = distance(pts[0],pts[2]);
    sides[2] = distance(pts[0],pts[1]);
}
```

(e) Write the getter functions for the `Triangle` class' `sides`.
ANS:

```
double getS1() { return sides[0]; }
double getS2() { return sides[1]; }
double getS3() { return sides[2]; }
```