1. (a) 0 1 2

   0 0 1 1 0 1 2 0 1

   True
   True
   False

(b) A function to generate the Fibonacci sequence:

```python
def fibo():
x = 0
y = 1
while 1:
    yield x
    x, y = y, x+y
```

It can be tested using:

```python
for a in fibo():
    print a
    if a > 30:
        break
```

2. (a) The dictionary data type or a class are two good options. The dictionary is suitable for storing exactly the given data. By defining a class more properties can be added.

(b) Appropriate: e.g. prototyping, top-level control programs
    Not appropriate: e.g. safety-critical systems, time-critical modules

(c) Regular expressions are string matching patterns, that can be used for searching, replacing and extracting information in text data.

3. #include <stdio.h>

    /*
     * function that tests if n is a perfect number
     * n, a positive integer
     * result, 1 if n is a perfect number, 0 otherwise
     */

    int perfect(int n) {
        /*
            sum keeps track the sum of all factors of n
        */
stop give us the biggest possible factor
*/

int sum = 0;
int stop = n/2;
int i;

/* test all numbers up to the maximum possible
   to find out which are evenly divisible with n
   if it is, add it to sum
*/

for (i = 1; i <= stop; i++)
   if (n % i == 0) sum = sum + i;

/* test and return result */

if (sum == n ) return 1;
else return 0;
}

int main() {
  int i;

  /* test all numbers up to one million, print all
     perfect numbers
*/

  for (i = 1; i <= 1000000; i++)
     if (perfect(i))
       printf("%d\n", i);
  return 0;
}

/*
output of this will be

tragula.it.uu.se> gcc perfekt.c
tragula.it.uu.se> a.out
6
28
496
8128
*/
4. #include <stdio.h>
   #include <assert.h>

/*
   compute the dotproduct of the arrays x and y
   parameters:
   x, y : the arrays
   xn, yn : the size of x and y
   result: if the arrays has the same size the dot
           product is returned otherwise the program
           is terminated
*/

double dotproduct(double x[], double y[], int xn, int yn) {

/*
   There is no way to find out the size from the array itself
   so it has to be provided separately
   Check that we have the same size. Here we use assert. This is
   a way to test things that should be true. If they are not
   the program is terminated.
*/

   assert(xn == yn);

   /* create a sum and set it zero */

double sum = 0;

   /* iterate over the arrays */

   int i;
   for (i = 0; i < xn; i++)
       sum += x[i]*y[i];

   /* return result */

   return sum;
}

/* create three array to test the above*/
int main(){
    double x[5] = {1,2,3,4,5};
    double y[5] = {-5, -4, -3, -2, -1};
    double z[6] = {1, 2,3,4,5,6};

    printf("dot product is %5.2f\n", dotproduct(x,y,5,5));

    /* this should fail */
    printf("dot product is %5.2f\n", dotproduct(x,z,5,6));
    return 0;
}

/*
output will be

tragula.it.uu.se> gcc dot.c
tragula.it.uu.se> a.out
dot product is -35.00
Assertion failed: xn == yn, file dot.c, line 5
Abort

*/

5. #include <iostream>
    #include <cmath>

    using namespace std;

    class Sphere {
    private:
        double x,y,z;
        double r;

    public:

        // parameterless constructor,
        // initialize x,y,z using an initializer
        Sphere():x(0),y(0),z(0),r(1) {}  

        // constructor, initialize x,y,z using assignments, we
could have used an initializer here also, but we show
an alternative. since the parameters and the attributes
have the same name the this pointer has to be used.

Sphere(double x, double y, double z, double r) {
    this->x = x;
    this->y = y;
    this->z = z;
    this->r = r;
}

// print a description of the object
void print() {
    cout << "Position is: " << "(" << x << " , " << y << " , " << z << ")" << endl
        << "Radius is: " << r << endl;
}

// calculate the volume, M_PI is defined in cmath
double volume() {
    return 4*M_PI*r*r*r/3;
}

// test if two spheres collide. One of the spheres
// is the current object, the other one is the parameter

bool collide(const Sphere& other) {

    // calculate euclidian distance between centre points
    double dist = sqrt((x-other.x)*(x-other.x) +
                        (y-other.y)*(y-other.y) +
                        (z-other.z)*(z-other.z));

    // check if close enough
    if(dist <= r + other.r) return true;
    else return false;
}

// create two spheres and check whether they collide, print
// result
int main() {
    Sphere a(1,1,2,1.5), b(1,1,3,0.2);
    a.print();
    b.print();
    cout << a.collide(b) << endl; // print 1 or 0
    cout << boolalpha << a.collide(b) << endl; // print true or false
    return 0;
}

/*
output will be
tragula.it.uu.se> g++ Sphere.cc
tragula.it.uu.se> a.out
Position is: (1, 1, 2)
Radius is: 1.5
Position is: (1, 1, 3)
Radius is: 0.2
1
true
*/

6. (a) If all messages have the same priority a first-in first-out (FIFO) queue is appropriate. The STL type queue provides such a structure.
(b) To traverse the set of elements in any STL collection container.
(c) template <class T>
    class Graph {
        struct Node {
            T data;
            vector<Node*> neighbors;
        };
        vector<Node> nodes;
    };

7. public class Sphere {
    // attributes as in the text
    private double x,y,z;
    private double r;

    // parameterless constructor, initialize all attributes
public Sphere() {
    x = y = z = 0;
    r = 1;
}

// other constructor, initialize x,y,z and r from the
// given parameters. Since we use the same names
// (we don’t need to) this must be used
public Sphere(double x, double y, double z, double r) {
    this.x = x;
    this.y = y;
    this.z = z;
    this.r = r;
}

// return a string that describes the object
public String toString() {
    return "Position is: (" + x + ", " + y + ", " + z + ")\n" + "Radius is: " + r;
}

// calculated the volume
double volume() {
    return 4*Math.PI*r*r*r/3;
}

// test if the current object collides with another
// object given as a parameter
boolean collide(Sphere other) {

    // calculate euclidian distance between centre points
    double dist = Math.sqrt((x-other.x)*(x-other.x) +
                            (y-other.y)*(y-other.y) +
                            (z-other.z)*(z-other.z));

    // test if we are close enough
    if(dist <= r + other.r) return true;
    else return false;
}
public static void main(String[] args) {
    Sphere a, b;
    a = new Sphere(1,1,2,1.5);
    b = new Sphere(1,1,3,0.2);
    System.out.println(a);
    System.out.println(b);
    System.out.println(a.collide(b));
}

/*
output will be
tragula.it.uu.se> javac Sphere.java
tragula.it.uu.se> java Sphere
Position is: (1.0, 1.0, 2.0)
Radius is: 1.5
Position is: (1.0, 1.0, 3.0)
Radius is: 0.2
true
*/

public class Sample {
    // our two arrays, in Java we know the size so we don't
    // need a size although there are sometimes a good reason
    // to have one

    private double[] x,y;

    // exception of our own

    private static class SizeError extends RuntimeException {
        public SizeError(String msg) {
            super(msg);
        }
    };

    // constructor giving us the two arrays. Be sure to copy these
    // do NOT just copy the references
public Sample(double[] xp, double[] yp) {

    // check that x and y have the same size, if not
    // produce an exception

    if (xp.length != yp.length)
        throw new SizeError("Arrays must have the same size");

    // create two arrays and copy parameters into them

    x = new double[xp.length];
    for (int i = 0; i < x.length; i++)
        x[i] = xp[i];
    y = new double[yp.length];
    for (int i = 0; i < y.length; i++)
        y[i] = yp[i];

}

// sum of all x’s. The method is private since there is no
// reason to call it from the outside

private double xsum() {
    double sum = 0;
    for(int i = 0; i < x.length; i++)
        sum += x[i];
    return sum;
}

// sum of all y’s

private double ysum() {
    double sum = 0;
    for(int i = 0; i < y.length; i++)
        sum += y[i];
    return sum;
}

// sum of all squared x’s

private double x2sum() {
    double sum = 0;
    for(int i = 0; i < x.length; i++)
        sum += x[i]*x[i];
}
return sum;
}

// dot product of x and y
private double xysum() {
    double sum = 0;
    for(int i = 0; i < x.length; i++)
        sum += x[i]*y[i];
    return sum;
}

// calculate a and b. Since we cannot return two things
// we pack them in an array that we return
double [] least() {
    // get the size of x
    int n = x.length;

    // calculate D, a and b using the methods previously defined
    double D = n*x2sum() - xsum()*xsum();
    double a = (n*xysum() - xsum()*ysum())/D;
    double b = (x2sum()*ysum() - xysum()*xsum())/D;

    // create a result array
    double[] result = new double[2];
    result[0] = a;
    result[1] = b;

    // return it
    return result;
}

public static void main(String [] args) {
    // create two arrays
    double[] x = {1, 2, 3, 4, 5};
    double[] y = {1.1, 1.96, 3.05, 3.97, 4.97};
// create an object

Sample s = new Sample(x, y);

// call the requested method

double[] res = s.least();

// print the result

System.out.println(res[0] + "* x + " + res[1]);

// formatted printout

System.out.format("%5.2f*x + %5.2f\n", res[0], res[1]);

} }

/*
output will be

tragula.it.uu.se> javac Sample.java
tragula.it.uu.se> java Sample
0.9749999999999989* x + 0.08500000000000227
 0.97*x + 0.09

*/

9.

10.