CSCI 135 Software Design and Analysis
Example classes and programs
we have seen in class

```
#include <iostream>
#include <cstdlib>

using std::cout;

class Matrix {  //a 2x2 matrix
  double a[4];

 public:
  Matrix() {
    set(1, 0, 0, 1); //identity matrix
  }

  Matrix(double a11, double a12, double a21, double a22) {
    set(a11, a12, a21, a22);
  }

  void set(double a11, double a12, double a21, double a22) {
    a[0]=a11;
    a[1]=a12;
    a[2]=a21;
    a[3]=a22;
  }

  double get(int i, int j) {  //assume 1<=i,j<=2
    return a[(i-1)*2+(j-1)];  //verify it is consistent with set
  }

  double det() {
    return a[0]*a[3]-a[1]*a[2]; //this is the determinant
  }

  bool singular() {
    return (det()==0);  //is the determinant zero
  }
```

```
  //review matrix multiplication rules
  //this is used like this: m3=m1.mul(m2);
  Matrix mul(Matrix m) {
    return Matrix(a[0]*m.a[0]+a[1]*m.a[2],
                  a[0]*m.a[1]+a[1]*m.a[3],
                  a[2]*m.a[0]+a[3]*m.a[2],
                  a[2]*m.a[1]+a[3]*m.a[3]);
  }

  //a matrix that when multiplied by this one gives the identity
  Matrix inverse() {
    double d=det();
    return Matrix(a[3]/d, -a[1]/d, -a[2]/d, a[0]/d);
  }

  void print() {
    cout<<'['<<a[0]<<','<<a[1]<<';'<<a[2]<<','<<a[3]<<']';
  }
};

int main() {
  //find the inverse of a random matrix
  Matrix p=Matrix(rand()%100, rand()%100, rand()%100, rand()%100);
  if (!p.singular()) {
    Matrix q=p.inverse();
    Matrix I=p.mul(q);
    I.print();
    cout<<'\n';
  }

  //generate Fibonacci sequence
  Matrix m=Matrix(0,1,1,1);
  Matrix r=m;
  int n=30;
  cout<<m.get(1,1)<<'\n';
  for (int i=1; i<n; i=i+1) {
    m=m.mul(r); //multiply by r repeatedly
    cout<<m.get(1,1)<<'\n';
  }
}
```

```
class Interval {
  int low;
  int high;

 public:
  Interval() { //empty interval
    set(0,-1);
  }

  Interval(int a) {
    set(a,a);
  }

  Interval(int l, int h) {
    set(l,h);
  }

  void set(int l, int h) {
    low=l;
    high=h;
  }

  int length() {
    return max(high-low+1, 0); //if <=0 (empty) return 0
  }

  bool empty() {
    return (low>high);
  }

  Interval intersect(Interval I) {
    return Interval(max(low, I.low), min(high, I.high));
  }

  void print() {
    if (empty())
      cout<<"[0,-1]";
    else
      cout<<'['<<low<<','<<high<<']';
  }

  //loop utility
  //to be used like this for example
  //Interval I=...
  //for (I.init(); !I.done(); I.advance(...)) {
  //  ...
  //  use I.get() in some way
  //  ...
  //}
 private:
  int counter;
```

```
 public:
  void init() {
    counter=low;
  }

  void advance(int i) {
    counter=counter+i;
  }

  bool done() {
    return (counter>high);
  }

  int get() {
    return counter;
  }
};

//suggested exercise
//given an array of interval
//determine if there are inclusion:
//[a,b] and [c,d] such that a<c<=d<b

bool inclusion(Interval a[], int n) {
  bool found=false;
  for (int i=0; i<n; i=i+1) {
    a[i].init();
    int l1=a[i].get();
    int h1=l1+a[i].length()-1;
    for (int j=i+1; j<n; j=j+1) {
      a[j].init();
      int l2=a[j].get();
      int h2=l2+a[j].length()-1;
      if (l1<l2 && l2<=h2 && h2<h1) {
        found=true;
        break;
      }
    }
    if (found)
      break;
  }
  return found;
}
//another way would be to sort the non-empty
//intervals by low end, and then check if high
//end is also sorted. In this case, there are
//no inclusions.
```

```
//an exercise in loop skips

//Assume we want to output the product
//of all numbers of an array except
//a[i], and we want to do it for all i.

//One way is to multiply all the numbers
//and then divide by a[i] for every i.

double p=1;
for (int i=0; i<n; i=i+1)
  p=p*a[i];
for (int i=0; i<n; i=i+1)
  cout<<p/a[i]<<'\n';

//this approach is efficient (no nested loops)
//but has a problem: if a[i] is 0 for some i,
//we can a run time error.
//this can be avoided by multiplying all numbers
//except zeros, and count the number of zeros.

double p=1;
int zeros=0;
int index=0;
for (int i=0; i<n; i=i+1)
  if (a[i]!=0)
    p=p*a[i];
  else {
    zeros=zeros+1;
    index=i;
  }

if (zeros==0) //no zeros, do it as before
  for (int i=0; i<n; i=i+1)
    cout<<p/a[i]<<'\n';

if (zeros>1) //all products are zeros
  for (int i=0; i<n; i=i+1)
    cout<<0<<'\n';

if (zeros==1) { //all products but one are zeros
  for (int i=0; i<index; i=i+1)
    cout<<0<<'\n';
  cout<<p<<'\n';
  for (int i=index+1; i<n; i=i+1)
    cout<<0<<'\n';


//To make this more interesting, assume
//that we are not allowed to use division
//now we have to explicitly skip an element
//each time from the product
```

```
for (int skip=0; skip<n; skip=skip+1) {
  double p=1;
  for (int i=0; i<n; i=i+1)
    if (i!=skip)
      p=p*a[i];
  cout<<p<<'\n';
}

//we pay the price of a doubly nested loop
//but there is a way to avoid this, it will
//be a nice puzzle to think about.
```