Object Oriented Programming
Object Oriented Programming

- For more structured code
struct Rational {

};
structured Rational {
    int num, denom;
};
struct Rational {
    int num, denom;
};

void write(Rational r, ostream& outs) {
}

struct Rational {
    int num, denom;
};

void write(Rational r, ostream& outs) {
    outs << r.num << "/" << r.denom;
}
struct Rational {
    int num, denom;
};

void read(Rational& r, istream& ins) {
    char slash;
    ins >> r.num;
    ins >> slash;
    ins >> r.denom;
}
struct Rational {
    int num, denom;
};

void read(Rational& r, istream& ins) {

    ins >> r.num;
    ins >> r.denom;
}
struct Rational {
    int num, denom;
};

void read(Rational& r, istream& ins) {
    char slash;
    ins >> r.num;
    ins >> slash;
    ins >> r.denom;
}
struct Rational {
    int num, denom;
};

mul(Rational r1, Rational r2) {
    Rational r;
    r.num = r1.num * r2.num;
    r.denom = r1.denom * r2.denom;
    return r;
}
struct Rational {
    int num, denom;
};

Rational mul(Rational r1, Rational r2) {
}

struct Rational {
    int num, denom;
};

Rational mul(Rational r1, Rational r2) {
    Rational r;
    r.num = r1.num * r2.num;
    r.denom = r1.denom * r2.denom;
    return r;
}
struct Rational {
    int num, denom;
};

Rational mul(Rational r1, Rational r2) {
    Rational r;
    r.num = r1.num * r2.num;
    r.denom = r1.denom * r2.denom;
    return r;
}
struct Rational {
    int num, denom;
};

Rational add(Rational r1, Rational r2) {
}
struct Rational {
    int num, denom;
};

Rational add(Rational r1, Rational r2) {
    Rational r;
    r.num    = r1.denom * r2.num + r2.denom * r1.num;
    r.denom  = r1.denom * r2.denom;
    return r;
}
struct Rational {
    int num, denom;
};

Rational add(Rational r1, Rational r2) {
    Rational r;
    r.num = r1.denom * r2.num + r2.denom * r1.num;
    r.denom = r1.denom * r2.denom;
    return r;
}
struct Rational {
    int num, denom;
};

Rational add(Rational r1, Rational r2) {
    Rational r;
    r.num = r1.denom * r2.num + r2.denom * r1.num;
    r.denom = r1.denom * r2.denom;
    return r;
}
struct Rational {
    int num, denom;
};

void reduce(Rational& r) {
    int g = gcd(r.num, r.denom);
    r.num /= g;
    r.denom /= g;
}

struct Rational {
    int num, denom;
};

void reduce(Rational& r)
{
}
```c
struct Rational {
    int num, denom;
};

void reduce(Rational& r)
{
    int g = gcd(r.num, r.denom);

    r.num /= g;
    r.denom /= g;
}
```
struct Rational {
    int num, denom;
};

void write(Rational r, ostream& outs);
void read(Rational& r, istream& ins);
Rational mul(Rational r1, Rational r2);
Rational add(Rational r1, Rational r2);
void reduce(Rational& r);
class Rational {
    int num, denom;

    void write(Rational r, ostream& outs);
    void read(Rational& r, istream& ins);
    Rational mul(Rational r1, Rational r2);
    Rational add(Rational r1, Rational r2);
    void reduce(Rational& r);
};
class Rational {
    int num, denom;

    void write(Rational r, ostream& outs);
    void read(Rational& r, istream& ins);
    Rational mul(Rational r1, Rational r2);
    Rational add(Rational r1, Rational r2);
    void reduce(Rational& r);
};
class Rational {
    int num, denom;

    void write(ostream& outs);
    void read(istream& ins);
    Rational mul(Rational r);
    Rational add(Rational r);
    void reduce();
};
class Rational {
    void write(ostream& outs);
    void read(istream& ins);
    Rational mul(Rational r);
    Rational add(Rational r);
    void reduce();

    int num, denom;
};
Object Oriented Programming

- Helps keep data and code together
class Rational {

    void write(ostream& outs);
    void read(istream& ins);
    Rational mul(Rational r);
    Rational add(Rational r);
    void reduce();

    int num, denom;
};
class Rational {
public:
    void write(ostream& Outs);
    void read(istream& Ins);
    Rational mul(Rational R r);
    Rational add(Rational R r);
    void reduce();

private:
    int num, denom;
};
Encapsulation

- Hide data from code that is outside the object.
- Only member functions can directly access data.
Information Hiding

Hide class implementation details from client.

This is called information hiding.

Public functions of a class provide the interface between the client and the class objects.
Information Hiding

You can drive a car without knowing how the engine works.
You can drive a car without knowing how the engine works.

You can use an object without knowing its implementation.
Object Oriented Programming

- Helps keep data and code together
Object Oriented Programming

- Helps keep data and code together
- Helps separate interface from implementation
Code that manipulates that data
Code that manipulates that data

Data

Interface

Code that manipulates that data
Object Oriented Programming

- Helps keep data and code together
- Helps separate interface from implementation
Object Oriented Programming

- Helps keep data and code together
- Helps separate interface from implementation
- Helps avoid code duplication (inheritance)
class Rational {
public:
    void write(ostream& outs);
    void read(istream& ins);
    Rational mul(Rational r);
    Rational add(Rational r);
    void reduce();

private:
    int num, denom;
};
Rational r1, r2;

Rational r3 = r1.add(r2);
r3.write(cout);
r3.reduce();
r3.write(cout);
Rational r1, r2;

r1.read(cin);
r2.read(cin);
Rational r1, r2;

r1.read(cin);
r2.read(cin);

Rational r3 = r1.add(r2);
Rational r1, r2;
r1.read(cin);
r2.read(cin);
Rational r3 = r1.add(r2);
r3.write(cout);
Rational r1, r2;

r1.read(cin);
r2.read(cin);

Rational r3 = r1.add(r2);

r3.write(cout);
r3.reduce();
r3.write(cout);
Rational r1, r2;

r1.read(cin);
r2.read(cin);

Rational r3 = r1.add(r2);

r3.write(cout);
r3.reduce();
r3.write(cout);

cout << r3.num;
Rational r1, r2;

r1.read(cin);
r2.read(cin);

Rational r3 = r1.add(r2);

r3.write(cout);
r3.reduce();

r3.write(cout);

cout << r3.num;
class Rational {
  public:
    void write(ostream& outs);
    void read(istream& ins);
    Rational mul(Rational r);
    Rational add(Rational r);
    void reduce();

  private:
    int num, denom;
};
void Rational::write(ostream& outs)
{
outs << num << "/" << denom;
}

void Rational::write(ostream& outs)
{
}

void Rational::write(ostream& outs) {
    outs << num << "/" << denom;
}
void Rational::read(istream& ins)
{
    char slash;
    ins >> num;
    ins >> slash;
    ins >> denom;
}
void Rational::read(istream& ins) {
    char slash;
    ins >> num;
    ins >> slash;
    ins >> denom;
}
void Rational::read(istream& ins) {
    char slash;
    ins >> num;
    ins >> slash;
    ins >> denom;
}
Rational Rational::mul(Rational r) {
}

Rational Rational::mul(Rational r) {
Rational Rational::mul(Rational r)
{
    Rational r2;
    r2.num    = num    * r.num ;
    r2.denom  = denom  * r.denom;
    return r2;
}
Rational Rational::add(Rational r)
{
    Rational r2;
    r2.num = denom * r.num + denum * num;
    r2.denom = denom * denum;
    return r2;
}
Rational Rational::add(Rational r) {
    Rational r2;
    r2.num = denom * r.num + r.denom * num;
    r2.denom = denom * r.denom;
    return r2;
}
```cpp
void Rational::reduce()
{
    int g = gcd(num, denom);

    num /= g;
    denom /= g;
}
```
Objects and Classes
Objects and Classes

- Classes reflect concepts
- Objects reflect instances that embody those concepts.

**object**

- Jodie
- Daria
- Jane
- Brittany

**class**

- girl
Objects and Classes

class BankAccount

    void deposit (int amount);
    void withdraw(int amount);

private:

    int balance;

balance 500
balance 10,000
Objects as instances of Classes

- The world conceptually consists of objects
- Many objects are of the same type or class
  - My bank account, your bank account, Bill Gates’ bank account
Class Instance Diagrams

Rational r1
- read
- write
- mul
- add
- reduce

Private data:
- num: 10
- denom: 13

Rational r2
- read
- write
- mul
- add
- reduce

Private data:
- num: 20
- denom: 15
Structs

- Typically all members public
- No member functions

Classes

- Typically all data members private
- Interface member functions public

Technically, same

- Perceptually, very different mechanisms
class Rational
{
    public:
        void read(istream&);
        void write(ostream&);
        Rational mul(Rational);
        Rational add(Rational);
        void reduce();
    private:
};
class Rational
{
    public:
        void read(istream&);
        void write(ostream&);
        Rational mul(Rational);
        Rational add(Rational);
        void reduce();
    private:
        int num, denom;
};
class Rational
{
    public:
        void read(istream&);
        void write(ostream&);
        Rational mul(Rational);
        Rational add(Rational);
        void reduce();
    private:
        int num, denom;
};
```cpp
class Rational
{
    public:
    void read(istream&);
    void write(ostream&);
    Rational mul(Rational);
    Rational add(Rational);
    void reduce();

    private:
    int num, denom;
};

void Rational::read(istream& ins)
{
    char slash;
    ins >> num >> slash >> denom;
}

...
class Rational
{
    public:
        void read(istream&);
        void write(ostream&);
        Rational mul(Rational);
        Rational add(Rational);
        void reduce();
    private:
        int num, denom;
};

void Rational::read(istream& ins)
{
    char slash;
    ins >> num >> slash >> denom;
}

...
Rational r1, r2;

r1.read(cin);
r2.read(cin);

Rational r3 = r1.add(r2);
r3.reduce();
r3.write(cout);

class Rational
{
    public:
        void read(istream&);
        void write(ostream&);
        Rational mul(Rational);
        Rational add(Rational);
        void reduce();
    private:
        int num, denom;
};

void Rational::read(istream& ins)
{
    char slash;
    ins >> num >> slash >> denom;
}

...
class Rational
{
    public:
        void read(istream&);
        void write(ostream&);
        Rational mul(Rational);
        Rational add(Rational);
        void reduce();
    private:
        int num, denom;
};

Rational r1, r2;
r1.read(cin);
r2.read(cin);
Rational r3 = r1.add(r2);
r3.reduce();
r3.write(cout);

void Rational::read(istream& ins)
{
    char slash;
    ins >> num >> slash >> denom;
}

...
class Rational
{
    public:
        void read(istream&);
        void write(ostream&);
        Rational mul(Rational);
        Rational add(Rational);
        void reduce();
    private:
        int num, denom;
};

#include "Rational.h"
Rational r1, r2;
r1.read(cin);
r2.read(cin);
Rational r3 = r1.add(r2);
r3.reduce();
r3.write(cout);

#include "Rational.h"

void Rational::read(istream& ins)
{
    char slash;
    ins >> num >> slash >> denom;
}

...
Separate Compilation and Linking of Files

interface file

Rational.h
Separate Compilation and Linking of Files

interface file

Rational.h

implementation file

Rational.cpp

#include "Rational.h"
Separate Compilation and Linking of Files

```
#include "Rational.h"
```

main program

Rational.h

Interface file

Rational.cpp

Implementation file

client.cpp
Separate Compilation and Linking of Files

- **client.cpp**
- **Rational.h**
- **Rational.cpp**
- **client.obj**

Compiler

#include "Rational.h"

interface file

implementation file
Separate Compilation and Linking of Files

- **client.cpp**
- **Rational.h**
- **client.obj**
- **Rational.cpp**
- **Rational.obj**

- **Compiler**
- **#include "Rational.h"**

- **main program**
- **interface file**
- **implementation file**
Separate Compilation and Linking of Files

- **client.cpp**
  - Compiler
  - client.obj

- **Rational.h**
  - interface file
  - Compiler
  - Rational.cpp
  - #include "Rational.h"
  - Rational.obj

- **Linker**
  - client.exe

- **main program**
  - implementation file

- **client.exe**
  - Compiler
  - Rational.obj
class Rational {
    public:
         void write(ostream& outs);
    void read(istream& ins);
    Rational mul(Rational r);
    Rational add(Rational r);
    void reduce();

    private:
         int num, denom;
};
class Rational {
    public:
    void write(ostream& outs);
    void read(istream& ins);
    Rational mul(Rational r);
    Rational add(Rational r);
    void reduce();

    private:
    int num, denom; // instance variables or fields
};
class Rational {
  public:
    void write(ostream& outs);
    void read(istream& ins);
    Rational mul(Rational r);
    Rational add(Rational r);
    void reduce();

  private:
    int num, denom;
};
void Rational::write(ostream& outs)
{
    outs << num << "/" << denom;
}
void Rational::write(ostream& outs) {
    outs << this.num << "/" << this.denom;

    // num is equivalent to this.num
    // denom is equivalent to this.denom
void Rational::write(Rational& this, ostream& outs) {
    outs << this.num << "/" << this.denom;
    // this is passed as a hidden argument
}
Implementation: What really happens

```c++
void Rational::write(Rational& this, ostream& outs) {
    outs << this.num << "/" << this.denom;
    // this is passed as a hidden argument
}
```
void Rational::write(ostream& outs)
{
    outs << this.num << "/" << this.denom;

    // this is passed as a hidden argument
}
void Rational::write(ostream& outs) const
{
    outs << this.num << "/" << this.denom;

    // this is passed as a hidden argument

    // const specifies fields of this can't be modified
}
void Rational::write(ostream& outs) const
{
    outs << num << "/" << denom;

    // this is passed as a hidden argument
    // const specifies fields of this can't be modified
    // num is equivalent to this.num
    // denom is equivalent to this.denom
}
void Rational::write(ostream& outs) const
{
    outs << num << "/" << denom;
}
class Rational {
  public:
    void write(ostream& outs);
    void read(istream& ins);
    Rational mul(Rational r);
    Rational add(Rational r);
    void reduce();
  
  private:
    int num, denom;
};
class Rational {
    public:
    void write(ostream& outs) const;
    void read(istream& ins);
    Rational mul(Rational r);
    Rational add(Rational r);
    void reduce();

    private:
    int num, denom;
};
class Rational {
    public:
    void write(ostream& outs) const;
    void read(istream& ins);
    Rational mul(Rational r) const;
    Rational add(Rational r) const;
    void reduce();

    private:
    int num, denom;
};
Rational Rational::mul(Rational r) const
{
    Rational r2;
    r2.num   = num * r.num ;
    r2.denom = denom * r.denom;
    return r2;
}
Rational Rational::add(Rational r) const
{
    Rational r2;
    r2.num = denom * r.num + r.denom * num;
    r2.denom = denom * r.denom;
    return r2;
}
class Rational
{
    public:
        void read(istream&);
        void write(ostream&);
        Rational mul(Rational);
        Rational add(Rational);
        void reduce();
    private:
        int num, denom;
};

#include "Rational.h"

Rational r1, r2;
r1.read(cin);
r2.read(cin);
Rational r3 = r1.add(r2);
r3.reduce();
r3.write(cout);

#include "Rational.h"

void Rational::read(istream& ins)
{
    char slash;
    ins >> num >> slash >> denom;
}

...
class Rational
{
    ...
};
```cpp
#include "Rational.h"

class Geometry
{
    ...
};

#include "Rational.h"

class Algebra
{
    ...
};
```

```cpp
class Rational
{
    ...
};
```
```cpp
class Rational
{
    ...
};

#include "Rational.h"

class Geometry
{
    ...
};

#include "Geometry.h"

class Algebra
{
    ...
};

#include "Algebra.h"

#include "Rational.h"

class Mathematics
{
    ...
};

#include "Geometry.h"
#include "Algebra.h"
```
class Rational
{
    ...
};

#include "Rational.h"

class Geometry
{
    ...
}

#include "Rational.h"

class Algebra
{
    ...
}

#include "Rational.h"

class Mathematics
{
    ...
}
class Rational {
    ...
};

class Geometry {
    ...
};

class Rational {
    ...
};

class Algebra {
    ...
};

class Mathematics {
    ...
};
Rational is included twice. This causes compile time error.
class Rational {
  public:
    void write(ostream& outs) const;
    void read(istream& ins);
    Rational mul(Rational r) const;
    Rational add(Rational r) const;
    void reduce();

  private:
    int num, denom;
};
#ifndef RATIONAL_H
#define RATIONAL_H

class Rational {
    public:
        void write(ostream& outs) const;
        void read(istream& ins);
        Rational mul(Rational r) const;
        Rational add(Rational r) const;
        void reduce();

    private:
        int num, denom;
};

#endif
Declaration + Initialization

```c
int x = 5;
int y(10);

bool flag(true);
```
int x = 5;
int y(10);

bool flag(true);

// How about:
Rational r(2,3);
Class Constructors

- The constructor is a member function
  - Used to initialize the data members of an object
The constructor is a member function
- Used to initialize the data members of an object

The name of a constructor is the name of the class

A constructor has no return type
class Rational {
    public:

    void write(ostream& outs) const;
    void read(istream& ins);
    Rational mul(Rational r) const;
    Rational add(Rational r) const;
    void reduce();

    private:
    int num, denom;
};
class Rational {

public:
    Rational();             // default constructor
    Rational(int n, int d); // another constructor
    void write(ostream& outs) const;
    void read(istream& ins);
    Rational mul(Rational r) const; // rational multiplication
    Rational add(Rational r) const; // rational addition
    void reduce();

private:
    int num, denom;
};
class Rational {
public:
    Rational(); // default constructor
    Rational(int n, int d); // another constructor
    void write(ostream& outs) const;
    void read(istream& ins);
    Rational mul(Rational r) const;
    Rational add(Rational r) const;
    void reduce();

private:
    int num, denom;
};
Rational::Rational()
{
}

Rational::Rational(int n, int d)
{
    num = n;
    denom = d;
}
Rational::Rational()
{
    num =
    denom =
}
Rational::Rational(int n, int d)
{  
    num = n;
    denom = d;
}
Rational::Rational()
{
    num    = 0;
    denom = 1;
}
Rational::Rational()
{
    num   = 0;
    denom = 1;
}

Rational::Rational(int n, int d)
{
    num   =
    denom =
}
Rational::Rational()
{
    num   = 0;
    denom = 1;
}

Rational::Rational(int n, int d)
{
    num   = n;
    denom = d;
}
class Rational {
    public:
        Rational(); // default constructor
        Rational(int n, int d); // another constructor
        void write(ostream& outs) const;
        void read(istream& ins);
        Rational mul(Rational r) const;
        Rational add(Rational r) const;
        void reduce();

    private:
        int num, denom;
};
Rational r1, r2(1,2), r3(1,3);

r1.write(cout);
Rational r1, r2(1,2), r3(1,3);
r1.write(cout);    // 0/1
r4 = r2.add(r3);
r4.write(cout);   // 5/6
Rational r1, r2(1,2), r3(1,3);

r1.write(cout); // 0/1
r2.write(cout);
Rational r1, r2(1, 2), r3(1, 3);

r1.write(cout); // 0/1
r2.write(cout); // 1/2
r3.write(cout); // 1/3
r4 = r2.add(r3);
r4.write(cout); // 5/6
Rational r1, r2(1,2), r3(1,3);

r1.write(cout); // 0/1
r2.write(cout); // 1/2
r3.write(cout);
Rational r1, r2(1,2), r3(1,3);

r1.write(cout); // 0/1
r2.write(cout); // 1/2
r3.write(cout); // 1/3
r4.write(cout); // 5/6
Rational r1, r2(1,2), r3(1,3);

r1.write(cout); // 0/1
r2.write(cout); // 1/2
r3.write(cout); // 1/3

Rational r4 = r2.add(r3);

r4.write(cout);
Rational r1, r2(1,2), r3(1,3);

r1.write(cout); // 0/1
r2.write(cout); // 1/2
r3.write(cout); // 1/3

Rational r4 = r2.add(r3);

r4.write(cout); // 5/6
#include <iostream>

char c;

cin >> c;    // cin is an instance of istream
cout << c;   // cout is an instance of ostream

cin.get(c);  // get(…) is a member function of istream
#include <fstream>

ifstream f;

f.open("data.txt"); // open(...) is a member function of fstream
#include <string>

string name = "eeecs183";

int l = name.length(); // length() is a member function of string
class Rational
{
    public:
        Rational();
        Rational(int n, int d);
        void write (ostream& outs) const;
        void read (istream& ins);
        Rational mul (Rational r) const;
        Rational add (Rational r) const;
        void reduce ();

    private:
        int num, denom;
};
class Stack
{
    public:

    private:

};
class Stack
{
    public:

    void push (int x);
    int pop ();

    private:

};
class Stack {
    public:
    Stack();
    void push (int x);
    int pop ();

    private:

    };

class Stack
{
public:
    Stack();
    void push (int x);
    int pop ();
    bool isEmpty () const;
    int size () const;
    void write (ostream& outs) const;

private:
};
class Stack
{
public:
    Stack();
    void push (int x);
    int pop ();
    bool isEmpty () const;
    int size () const;
    void write (ostream& outs) const;

private:
    int s[MAXSIZE];
    int top;
    int length;
};
Example:
MAXSIZE = 10;
top = 5;
length = 6;

| x | x | x | x | x | x |   |   |   |   |
|-----------------------------|
\begin{align*}
\text{top}
\end{align*}
Stack::Stack()
{

}

void Stack::push(int x)
{

}

int Stack::pop()
{

}
Stack::Stack()
{
}

void Stack::push(int x)
{
    top++;
    s[top] = x;
    length++;
}

int Stack::pop()
{
}
Stack::Stack()
{
}

void Stack::push(int x)
{
    top++;
    s[top] = x;
    length++;  
}

int Stack::pop()
{
    int x = s[top];
    top--;
    length--;
    return x;
}
Stack::Stack()
{
    length = 0;
    top = -1;
}

void Stack::push(int x)
{
    top++;  
    s[top] = x;  
    length++;  
}

int Stack::pop()
{
    int x = s[top];  
    top--;  
    length--;  
    return x;  
}
bool Stack::isEmpty() const
{
}

int Stack::size() const
{
}

void Stack::write(ostream& outs) const
{
}
bool Stack::isEmpty() const
{
    return (length == 0);
}

int Stack::size() const
{

}

void Stack::write(ostream& outs) const
{
}
bool Stack::isEmpty() const
{
    return (length == 0);
}

int Stack::size() const
{
    return length;
}

void Stack::write(ostream& outs) const
{
}
bool Stack::isEmpty() const
{
    return (length == 0);
}

int Stack::size() const
{
    return length;
}

void Stack::write(ostream& outs) const
{
    for (int i=0; i<length; i++)
    {
        outs << s[i] << " ";
    }
    outs << endl;
}
class Queue
{
    public:
        Queue();
        void enqueue(int x);
        int dequeue();
        bool isEmpty() const;
        int size() const;
        void write(ostream& outs) const;

    private:
        int q[MAXSIZE];
        int front;
        int back;
        int length;
};
class Queue
{
    public:
        void enqueue (int x);
        int dequeue ();

    private:

};
class Queue
{
public:
    Queue();
    void enqueue (int x);
    int dequeue ();
    int size () const;
    bool isEmpty () const;

private:
    int q[MAXSIZE];
    int front;
    int back;
    int length;
};
class Queue
{

public:
    Queue();
    void enqueue (int x);
    int dequeue ();
    bool isEmpty () const;
    int size () const;
    void write (ostream& outs) const;

private:

};
class Queue
{
    public:
        Queue();
        void enqueue (int x);
        int dequeue ();
        bool isEmpty () const;
        int size () const;
        void write (ostream& outs) const;
    
    private:
        int q[MAXSIZE];
        int front;
        int back;
        int length;
};
Example 1:
MAXSIZE = 10;
front = 1;
back = 5;
length = 5;

-----------------------------------------
|   | x | x | x | x | x |   |   |   |   |
-----------------------------------------
back               front

Example 2:
MAXSIZE = 10;
front = 8;
back = 3;
length = 6;

-----------------------------------------
| x | x | x | x |   |   |   |   | x | x |
-----------------------------------------
back               front
Queue.cpp

Example 1:
MAXSIZE = 10;
front = 1;
back = 5;
length = 5;

-----------------------------------------
|   | x | x | x | x | x |   |   |   |   |
-----------------------------------------

Example 2:
MAXSIZE = 10;
front = 8;
back = 3;
length = 6;

-----------------------------------------
| x | x | x | x | x |   |   |   | x | x |
-----------------------------------------
Queue::Queue() {
}

void Queue::enqueue(int x) {
}

int Queue::dequeue() {
}
Queue::Queue()
{
}

void Queue::enqueue(int x)
{
    back = ( back + 1 ) % MAXSIZE;
    q[back] = x;
    length++;
}

int Queue::dequeue()
{
}

Queue::Queue()
{

}

void Queue::enqueue(int x)
{
    back = ( back + 1 ) % MAXSIZE;
    q[back] = x;
    length++;
}

int Queue::dequeue()
{
    int x = q[front];
    front = ( front + 1 ) % MAXSIZE;
    length--;
    return x;
}
Queue::Queue()
{
    length = 0;
    front = 0;
    back = MAXSIZE-1;
}

void Queue::enqueue(int x)
{
    back = (back + 1) % MAXSIZE;
    q[back] = x;
    length++;
}

int Queue::dequeue()
{
    int x = q[front];
    front = (front + 1) % MAXSIZE;
    length--;
    return x;
}
bool Queue::isEmpty() const
{
}

int Queue::size() const
{
}

void Queue::write(ostream& outs) const
{
}
bool Queue::isEmpty() const
{
    return (length == 0);
}

int Queue::size() const
{
}

void Queue::write(ostream& outs) const
{
}
bool Queue::isEmpty() const
{
    return (length == 0);
}

int Queue::size() const
{
    return length;
}

void Queue::write(ostream& outs) const
{
    for (int i = 0; i < length; i++)
    {
        outs << q[(front + i) % MAXSIZE] << " ";
    }
    outs << endl;
}
bool Queue::isEmpty() const
{
    return (length == 0);
}

int Queue::size() const
{
    return length;
}

void Queue::write(ostream& outs) const
{
    for (int i=0; i<length; i++)
    {
        outs << q[(front + i) % MAXSIZE] << " ";
    }
    outs << endl;
}
class Set
{
    public:

    private:

};
class Set
{
    public:

    void insert (int x);
    void remove (int x);
    bool isMember (int x) const;

    private:

};
class Set
{
 public:
    Set();
    void insert (int x);
    void remove   (int x);
    bool isMember (int x) const;

 private:
};
class Set {
    public:
        Set();
        void insert (int x);
        void remove (int x);
        bool isMember (int x) const;
        bool isEmpty () const;
        int size () const;
        void write (ostream& outs) const;
        Set union (Set s);
        Set intersection (Set s);
    private:
};
class Set
{
    public:
        Set();
        void insert (int x);
        void remove (int x);
        bool isMember (int x) const;

        bool isEmpty () const;
        int size () const;

        void write (ostream& outs) const;

    private:
};
class Set
{
public:
    Set();
    void insert (int x);
    void remove (int x);
    bool isMember (int x) const;

    bool isEmpty () const;
    int size () const;

    void write (ostream& outs) const;

    Set union (Set s) const;
    Set intersection (Set s) const;

private:
};