

Abstract Data Type (ADT)

- **A data type**
- **Description**
 - Data components
 - Defined operations
- **Abstract**
 - Implementation not specified

Data Abstraction??

- **What's the point, anyway?**
- **First, what are we doing?**
 - Encapsulating (hiding) implementation
 - Publishing abstract definition
- **Why?**
 - Insulate ADT user from future changes
 - Simplify programmer's life!

Abstraction in Real Life

- **Simplify complex reality**
 - View at high level
 - Ignore low-level detail
 - Facilitate understanding
- **External appearance**
 - Hide internal operation
- **What if we didn't have it?**
 - Steering wheel does what??



ADT Advantages

- Intellectual manageability
- Security and integrity
- Sharing and re-use
- Maintainability



Maintainability Issues

$$q = p(n-1)$$

$$m = 1 + q + q^2 + q^3 + \dots$$

$$= \frac{1}{1-q}$$

n = number of modules in a system

p = probability that change in one module will cause change in another

m = number of changes resulting from single initial change

Courtesy Admiral Grace Hopper, USN (See text, page 455)

What if q = 1?
How do we reduce it?

The Bottom Line

Abstraction
is a **BIG** thing!

First Class Types - Operations

- Object definition/initialization
- Assignment
- “Arithmetic” operations
- “Relational” operations
- Input/output

Which Types are “First Class”?

	int	double	array	class
Definition	Y	Y	Y	Y
Assignment	Y	Y	N	Y
“Arithmetic”	Y	Y	N	?
“Relational”	Y	Y	N	?
Input/output	Y	Y	N?	?

Rational Number Example

- **Numeric types**
 - Integer: no fractional part
 - Real: fractional, but imprecise?
- **Rational number**
 - Value = numerator / denominator
 - Operations same as integer, real?

Similar (not identical) to book example

Rational Number ADT

- **Data**
 - Numerator, denominator (integers)
- **Operations**
 - Construct rational number object
 - Arithmetic: add, sub, mul, div
 - Relational: equal, less than
 - Input/output: insert, extract
 - Type conversion (later)

Rational Number Constructor

```
class Rational
{
public:
  Rational (int n=0, int d=1);
  ...
private:
  int num;
  int den;
};
```

What is the effect of default values?

Alternative(?):
Rational();
Rational(int n,int d=1);

Addition and Subtraction

```
class Rational
{
public:
  Rational Add (const Rational& right) const;
  Rational Subtract (const Rational& right) const;
  ...
};
```

Addition/subtraction member functions

Argument is "right" operand; object itself is "left" operand.

Doing Addition and Subtraction

```

Rational a (1,3); // 1/3
Rational b (1,6); // 1/6
Rational c;      // 0/1
Rational d;      // 0/1
c = a.Add (b);  ← Add "b" to "a"; return sum;
                  does not modify "a"
d = b.Subtract (a);

```

Same idea for multiplication/division

Relational Operations

```

class Rational
{
public:
...
bool Equal
  (const Rational& right) const;
bool LessThan
  (const Rational& right) const;
...
};

```

Similar to add/subtract, but return value is boolean.

Comparing Rational Numbers

```

Rational a (1,3); // 1/3
Rational b (1,6); // 1/6

if (a.Equal(b))
  ... ;
while (b.LessThan(a))
  ... ;

```

Do we need other comparisons?

I/O Operations

```

class Rational
{
public:
    Insert into output stream (const)
    void Insert
        (ostream& os) const;
    void Extract
        (istream& is);
};
    Extract from input stream (non-const!)

```

I/O Examples

```

Rational a (1,3); // 1/3
Rational b (1,6); // 1/6
a.Insert(cout) ← Format and output a's
                  value to stream
ifstream infile ("in.txt");
b.Extract(infile) ← Read characters from
                    file and set b's value

```

Accessors/Mutators

```

class Rational  What about accessibility?
                (public, private, protected?)
{
private:
    int GetNumerator() const;
    void SetNumerator(int n);
};
    Get or set numerator
    (same for denominator)

```
