# Friends of Classes
# Operator overloading

# Operator Overloading

- C++ allows overloading operators to work with classes
- What are some C++ operators?

https://en.cppreference.com/w/cpp/language/operators

# operator> *GreaterThan*

```
class SimpleRectangle // SimpleRectangle.h
{
  public:
    SimpleRectangle(double length, double width);
    bool operator>(const SimpleRectangle &rcRHS) const;

  private:
    double mLength, mWidth;
};

// SimpleRectangle.cpp
bool SimpleRectangle::operator>(const SimpleRectangle &rcRHS) const
{
  return mLength * mWidth > rcRHS.mLength * rcRHS.mWidth;
}
```

# Usage

```
int main()
{
  SimpleRectangle cBoxOne(100.0, 20.5);
  SimpleRectangle cBoxTwo(50.1, 25.3);

  if( cBoxOne > cBoxTwo )
  {
     cout << “BoxOne is bigger!”;
  }

  // ???? What happens here?
  if( cBoxOne > 100.0 )
  {
     cout << “BoxOne is bigger than 100.0!”;
  }
}
```

# operator> *GreaterThan*

```
class SimpleRectangle // SimpleRectangle.h
{
  public:
    SimpleRectangle(double length, double width);
    bool operator>(const SimpleRectangle &rcRHS) const;
    bool operator>(double area) const;

  private:
    double mLength, mWidth;
};

// SimpleRectangle.cpp
bool SimpleRectangle::operator>(double area) const
{
  return mLength * mWidth > area;
}
```

# Usage

```
int main()
{
  SimpleRectangle cBoxOne(100.0, 20.5);

  if( cBoxOne > 100.0 )
  {
     cout << “BoxOne is bigger than 100.0!”;
  }

  if( cBoxOne > 30 ) // ?????
  {
     cout << “BoxOne is bigger than 30!”;
  }

  if( 100 > cBoxOne ) // ?????
  {
     cout << “BoxOne is not greater than 100!”;
  }
}
```

# Problem!

- `if( 100 > cBoxOne )`

- An int is the first parameter
  - Can't be a member of Rectangle
  - Needs access to mLength and mWidth!

- Solution: `friend function`

# Friend

| SimpleRectangle |
| --- |
| -mLength : double<br>-mWidth : double |
| +SimpleRectangle(double, double)<br>+operator>(SimpleRectangle &) const : bool<br>+operator>(double) const : bool<br><br>«friend» operator«(double, SimpleRectangle&) : bool |

```
class SimpleRectangle // SimpleRectangle.h
{
  public:
    SimpleRectangle(double length, double width);
    bool operator>(const SimpleRectangle &rcRHS) const;
    bool operator>(double area) const;

    friend bool operator>(double area, const SimpleRectangle &rcRHS);
  private:
    double mLength, mWidth;
};

// SimpleRectangle.cpp
bool operator>(double area, const SimpleRectangle &rcRHS)
{
  return area > rcRHS.mLength * rcRHS.mWidth;
}
```

# Usage

```
int main()
{
  SimpleRectangle cBoxOne(100.0, 20.5);

  if( 101.5 > cBoxOne ) // ?????
  {
     cout << “BoxOne is not greater than 101.5!”;
  }

  if( 100 > cBoxOne ) // ?????
  {
     cout << “BoxOne is not greater than 100!”;
  }
}
```

# Friends of Classes

- A friend can be
  a) a regular stand-alone function
  b) a member of another class
  c) an entire class

# Overloading Stream Operators

- operator>>(istream, userObject)
  - Extraction operator
- operator<<(ostream, userObject)
  - Insertion operator
- Works with keyboard, screen, files, etc.
  - ostream and istream

# operator>>

```
class SomeClassName // SomeClassName.h
{
  public:
    friend istream& operator>> (istream &rcInput,
                                SomeClassName &rcObject);
  private:
    int mData;
};


// SomeClassName.cpp
istream& operator>> (istream &rcInput, ClassDef &rcObject)
{
  rcInput >> rcObject.mData;
  return rcInput;
}
```

# Overloading Stream Operators

- The general format for overloading the stream operators is as follows:

```
class SomeClassName
{
  public:
    .....
    friend istream& operator>> (istream &rcInput,
                                ClassDef &rcObject);
    friend ostream& operator<< (ostream &rcOutput,
                                const ClassDef &rcObject);
  private:
    .....
};
```

# Overload Insertion Operator <<

```
class Rational
{
  public:
    Rational (int = 0, int = 1);
    .....
    friend ostream& operator<< (ostream &rcOutput,
                                const Rational &rcRational);
  private:
    ...
};
```

# Overload Insertion Operator <<

```
ostream& operator<< (ostream &rcOutput,
                     const Rational &rcRational)
{
  rcOutput << rcRational.mNumerator << '/'
           << rcRational.mDenominator;

  return rcOutput;
}
```

# Overload Problem for Rational

1. Overload the insertion operator <<
2. Overload the extraction operator >>
3. Overload the multiplication operator *

   Rational operator*(const Rational &rcRHS) const;

4. Write a test driver