CSI33 Data Structures

Department of Mathematics and Computer Science
Bronx Community College
1 **CHAPTER 9: C++ CLASSES**

- Operator Overloading
- Class Variables and Methods
- In-class work
Overloading Operator Symbols

Overloading '++' as Standalone Function

class Rational
{
  public:
    Rational(int n = 0, int d = 1) { set(n, d); }

    // access functions
    int num() const { return num_; }
    int den() const { return den_; }

  private:
    int num_, den_;}

Rational operator+(const Rational &r1, const Rational &r2);
Overloading Operator Symbols

Overloading ‘+’ as Standalone Function

Rational operator+(const Rational &r1, const Rational &r2)
{
    int num, den;
    num = r1.num() * r2.den() + r1.den() * r2.num();
    den = r1.den() * r2.den();
    return Rational(num, den);
}

comments:
1) The function is not a member of the class Rational, so it cannot access the private data members directly
2) We pass the parameters as const reference parameters, i.e. only const methods of class Rational can be called, and only address of the objects is passed (less data is being copied - faster and uses less memory)
Overloading Operator Symbols

Overloading ‘+’ as Standalone Function

```cpp
// mainv1.cpp

#include "Rationalv1.h"

int main()
{
    Rational r1(2, 3), r2(3, 4), r3;
    r3 = r1 + r2; // common method of calling
    r3 = operator+(r1, r2); // direct method of calling
}
```

See Rationalv1.h, Rationalv1.cpp, and mainv1.cpp
Overloading Operator Symbols

Overloading '+' as Member Function

class Rational {
public:
    Rational(int n = 0, int d = 1) { set(n, d); }
    ...
    // access functions
    int num() const { return num_; }
    int den() const { return den_; }
    ...
    Rational operator+(const Rational &r2) const;
private:
    int num_, den_
};
... Rational Rational::operator+(const Rational &r2) const {
    Rational r;
    r.num_ = num_ * r2.den_ + den_ * r2.num_;
    r.den_ = den_ * r2.den_;
    return r;
}
...

See Rationalv2.h, Rationalv2.cpp, and mainv2.cpp
Overloading Operator Symbols

Overloading ‘+’ as Member Function

```cpp
// mainv2.cpp
#include "Rationalv2.h"
int main()
{
    Rational r1(2, 3), r2(3, 4), r3;
    r3 = r1 + r2; // common method of calling
    r3 = r1.operator+(r2); // direct method of calling
}
```
Overloading Operator Symbols

Overloading cin and cout

To override the cin and cout operators, they must be written as standalone functions, because they are not instances of the class Rational.

```cpp
std::istream& operator>>(std::istream &is, Rational &r) {
    char c;
    is >> r.num_ >> c >> r.den_;
    return is;
}
```

Comment: we have to put `std::` because we don’t do `using namespace std;` - a convention not to put it into header files.
Overloading Operator Symbols

**Friend Functions and Classes**
- Declared within the definition of a class using the keyword `friend`
- Allowed to have access to the private data and functions of the class.
- Needed for efficient performance with other classes.

**Friend Example: Rational.h**

```cpp
friend std::istream& operator>>(std::istream& is, Rational &r);
friend std::ostream& operator<<(std::ostream& os, const Rational &r);
...
std::istream& operator>>(std::istream &is, Rational &r);
std::ostream& operator<<(std::ostream &os, const Rational &r);
```
Overloading Operator Symbols

**Friend Example: Rational.cpp**

```cpp
std::istream& operator>>(std::istream &is, Rational &r) {
    char c;
    is >> r.num_ >> c >> r.den_;  
    return is;
}
std::ostream& operator<<(std::ostream &os, const Rational &r) {
    os << r.num() << "/" << r.den();
    return os;
}
```

See Rationalv3.h, Rationalv3.cpp, and mainv3.cpp
Class Variables

Syntax

- Declared using the `static` keyword.
- All instances (objects) in the class share the same value for a class variable. There is only one value for the entire class.
- Just as in the Python Card class.

Example: Card.h

```cpp
class Card {
    ...
private:
    ...
    static const std::string suits_[4];
    static const std::string faces_[13];
};
```
## Class Methods

### Syntax

- Declared using the `static` keyword.

- Can *only access class variables*. (A function call to a class method is not related to any particular instance.)

- Can be used to count how many instances are alive for a class: increment count in the constructor, decrement the count in the destructor.

- Must call using the class name and scope qualifier: `Card::count()`.

See `Card.h`, `Card.cpp`, `usingCard.cpp`, `test_Card.cpp`.
0. Look over the code of the **Card** class.

1. Add a static counter variable to the class Card (**static int count_;** in the **public** section)

2. In the implementation file (Card.cpp) add the initialization of the count_ variable: **int Card::count_=0;**

3. Increment the count_ variable in the constructor.

4. The things we did above will allow us to call the variable in our main function (**cout <<Card::count_ << endl;**)

5. The idea of making count_ variable public might be not be good. Think how we can fix it and do it.