Object Oriented Programming (CS1143)

Week 16

Department of Computer Science

Capital University of Science and Technology (CUST)

Outline

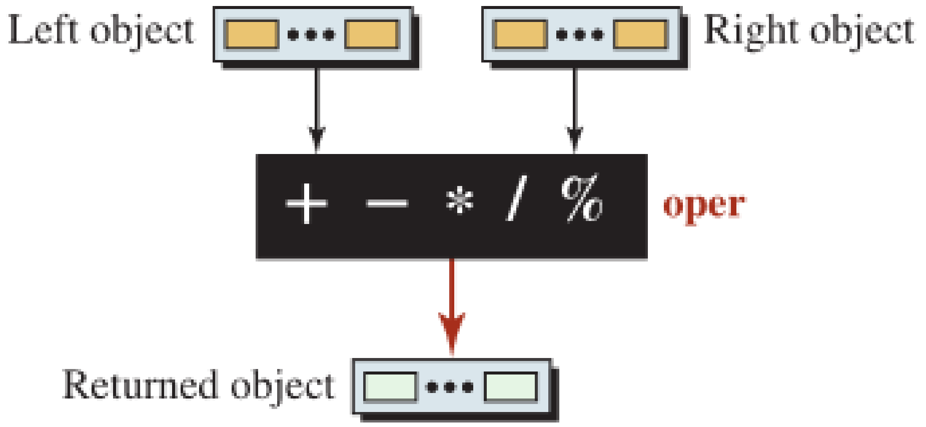
- Operator Overloading as a nonmember function
- Overloading binary arithmetic operator +
- Overloading stream extraction operator >>
- Overloading stream insertion operator <<

Overloading as a nonmember function

- When we overload a binary operator as a member function, one of the operands needs to be the host object. This is fine when each operand has a different role in the operation.
- However, in some operators, such as (a + b), the two operands play the same role and neither of them is related to the result. In these cases, it is better to use a nonmember function.
- C++ allows functions to be declared as friend functions of the class.
- A friend function has no host object, but it is granted friendship so that it can access the private data members and member functions of the class without calling the public member functions.
- We use friend functions to overload selected operators.

Binary Arithmetic Operators

- It is more appropriate to overload binary arithmetic operators as friend functions.
- The two operands must already exist. We create a new object inside the function and return it.
- We can pass the two operands as reference, but the return object cannot be a reference type because it is created inside the function definition.
- We can instantiate an object and then assign the results to that object if needed (fract = fract1 + fract2). We must make sure that the assignment operator is overloaded.



Prototye

```
friend const type operator oper (const type & left, const type & right)
```

```cpp
# include <iostream>
using namespace std;
class Fraction
{
    private:
        int numer;
        int denom;
    public:
        Fraction (int numer, int denom); // Parameter constructor
        friend Fraction operator+(const Fraction& left, const Fraction& right);
        void print();

};
```

```cpp
Fraction :: Fraction (int num, int den = 1)
: numer (num), denom (den)
{}

Fraction operator+ (const Fraction& left, const Fraction& right)
{
    int newNumer = left.numer * right.denom + right.numer * left.denom;
    int newDenom = left.denom * right.denom;
    Fraction result (newNumer, newDenom);

    return result;
}

void Fraction::print()
{
    cout<<numer<<"/"<<denom<<endl;
}
```

$$\frac{a}{b}+\frac{c}{d} \longrightarrow \frac{a*d+b*c}{b*d}$$

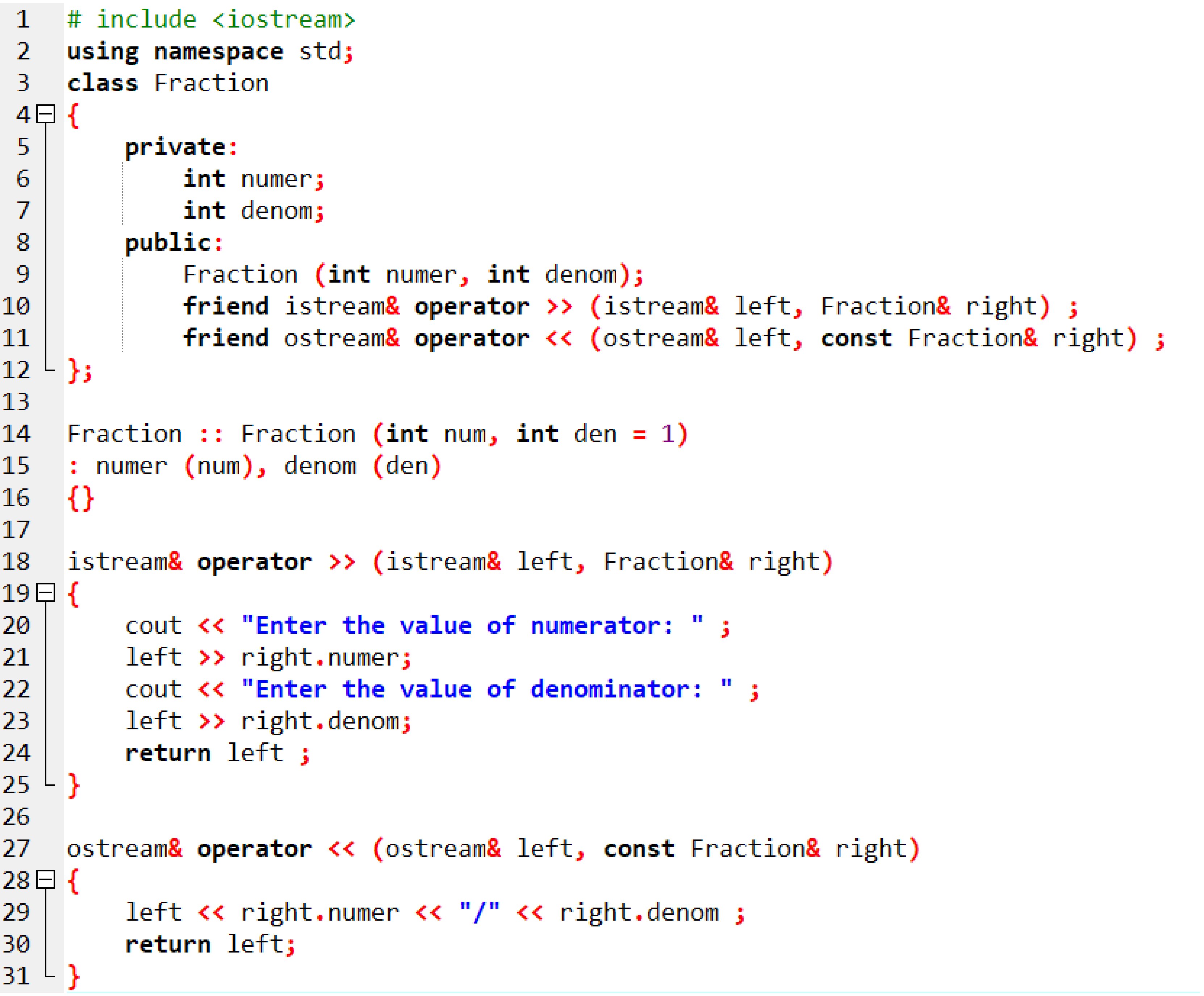
```cpp
int main ()
{
    Fraction fract1 (1,3);
    Fraction fract2 (1,2);

    (fract1+fract2).print();

    return 0;
}
```

Extraction and Insertion Operators

- The value of a fundamental type can be extracted from an input stream object using the extraction operator (>>) or inserted into an output stream using the insertion operator (<<).
- We can overload these two operators for our class types
- Each of these operators is a binary operator, but the left operand is an object of the istream class in the case of the extraction operator and the object of the ostream class in the case of the insertion operator.

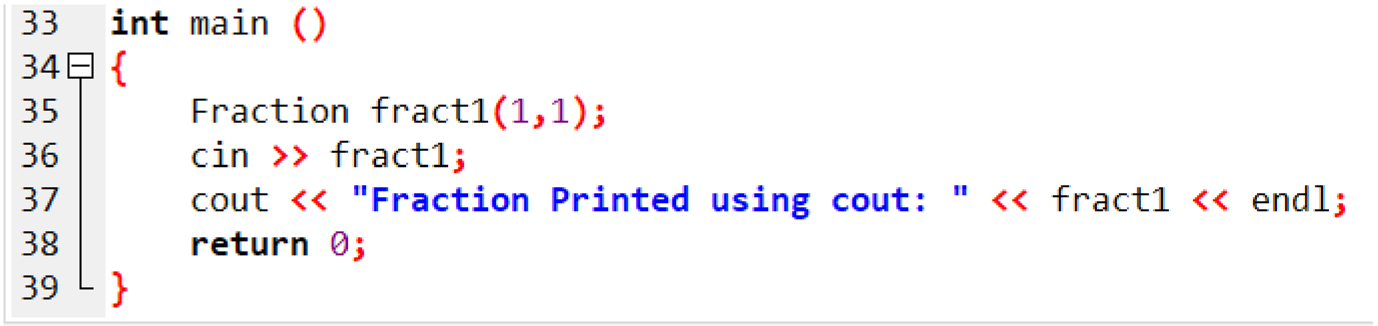


```cpp
# include <iostream>
using namespace std;
class Fraction
{
    private:
        int numer;
        int denom;
    public:
        Fraction (int numer, int denom);
        friend istream& operator >> (istream& left, Fraction& right) ;
        friend ostream& operator << (ostream& left, const Fraction& right) ;
};

Fraction :: Fraction (int num, int den = 1)
: numer (num), denom (den)
{}

istream& operator >> (istream& left, Fraction& right)
{
    cout << "Enter the value of numerator: " ;
    left >> right.numer;
    cout << "Enter the value of denominator: " ;
    left >> right.denom;
    return left ;
}

ostream& operator << (ostream& left, const Fraction& right)
{
    left << right.numer << "/" << right.denom ;
    return left;
}
```



```
int main ()
{
    Fraction fract1(1,1);
    cin >> fract1;
    cout << "Fraction Printed using cout: " << fract1 << endl;
    return 0;
}
```

This is all for Week 16