
Friends

Chapter 11.3

References in C++

```
struct Person
{
    char nameStr[20];
    char ssNum[9];
    int age;
};
```

- What do each of the following declarations mean?

```
Person sPersonStruct;
```

```
Person personArray[5];
```

```
Person *pPerson = &sPersonStruct;
```

```
Person &personRef = sPersonStruct;
```

References in C++

- `Person &personRef = personStruct;`
- A reference is like a constant pointer that is automatically dereferenced

```
int x = 0;
```

```
int &a = x;
```

```
cout << x << a << endl;
```

```
a++;
```

```
cout << x << a << endl;
```

Rules for References

- A reference must be initialized when it is created
- Once a reference is initialized to an object, it cannot be changed to refer to another object
- You cannot have NULL references

friend Functions

- Only the member functions of a class have direct access to the private data members of the class
- **friend** functions are friends of the class that are defined outside of the class but still have access to private data members

friend Functions

- The function prototype is placed in the class, preceded by the keyword **friend**
- The function definition can be written anywhere without the class name (class::)
- The function is able to directly access the private data members

friend Functions

- The `friend` function could be a member function in another class
- A class could also be made a friend of an existing class
 - In this case, every member function of the friend class will have access to this class's private data

Overloading Stream Operators

- Two classes named ostream and istream provide stream I/O.
- Definitions for >> and << are provided for the primitive datatypes such as int, float, char, and so on but not for user-defined types. These operators can be overloaded for our **Rational** class. As an example, we would like the following to have meaning:

```
Rational cR1(3,1);
```

```
cout << "Enter a rational number:";  
cin >> cR1;
```

In particular, we would like to be able to enter a value such as 1/3 for cR1.

Overloading the stream operators

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

```
class classDef
{
    public:
        .....
        friend ostream& operator>> (ostream& outputStream,
                                     classDef& variable);
        friend istream& operator<< (istream& inputStream,
                                     const classDef& variable);

    private:
        .....
};
```

Overloading the stream operators

- Note: For the stream extraction operator `>>` some `istream` object is passed to the operator function through **`istr`** such as `cin`.
- Similarly, the stream insertion operator `<<` is passed some `ostream` object through **`ostr`** such as `cout`.
- The function returns a modified stream so that the following chain can be executed correctly:
- `cin >> r1 >> r2`; Similar logic is used for the insertion operator `<<` function.

overload the insertion operator <<

```
class Rational
{
    private:
        ...
    public:
        Rational (void);
        Rational (int, int);
        .....
        friend ostream& operator<< (ostream &,
                                     const Rational &);
};
```

overload the insertion operator <<

```
ostream& operator<< (ostream& outputStream,  
                    const Rational& cRational)  
{  
    outputStream << cRational.numerator << '/'  
                << cRational.denominator;  
    return outputStream;  
}
```