Notes on overloading operators as `friends`

1. An operator that has different meanings with different data types is said to be overloaded.

2. Any function that overloads an operator is called an operator function.

3. The syntax of the heading of an operator function is:

   ```
   returnType operator operatorSymbol (parameters)
   ```

   E.g:

   ```
   istream& operator>> (istream& ins, RealNr& r)
   ```

   ```
   bool operator== (const RealNr& R1, const RealNr& R2)
   ```

   ```
   RealNr operator++ (RealNr& R1)
   ```

3. In C++, `operator` is a reserved word.

5. Operator functions always return a value. See highlighted statements in program, e.g.:

   ```
   RealNr operator ++ (RealNr& R1)
   {
      ++R1.intPart;
      return R1;
   }
   ```

6. A function call to an overloaded operator translates to the following:

   ```
   operatorSymbol (parameters)
   ```

   See e.g. the function call to the overloaded operator ++ in the code segment

   ```
   RealNr nr1;
   ++nr1
   ```

   In this case the **++** is the `operatorSymbol` and **`nr1`** is the parameter.

   Similarly in the function call:

   ```
   cin >> nr4;
   ```

   `cin` is the return value, the `>>` is the `operatorSymbol` and `nr4` is the parameter.

7. To use an operator on class objects, the operator must be overloaded. We then use the operator as it would have been used with pre-defined or built-in data types such as `int` or `double`.

8. When an operator is overloaded, its precedence cannot be changed, its associativity cannot be changed, default parameters cannot be used, the number of parameters the operator takes cannot be changed, and the way in which an operator works with built-in or pre-defined data types remains the same.

9. It is not possible to create new operators. Only existing operators can be overloaded.

10. A `friend` function is a non-member of a class, but has direct access to member variables.

11. The heading of the prototype of a `friend` function is preceded by the word `friend` in the header file, but omitted in the implementation file.

12. Since a `friend` function is not a member of the class, the scope of the class does not precede its implementation in the implementation file.

13. The operators that overload the stream insertion operator << and the stream extraction operator >> for a class must be `friend` functions.

14. In the attached example program, we overload both the stream insertion operator << and the stream extraction operator >>.

15. In the attached example program we also overload the operator+ as an example of overloading a binary operator. Other binary operators such as +, -, *, / can be overloaded and used in a similar way.

16. The operator++ is overloaded as an example of a unary operator. The operator -- can be overloaded in s similar way.

17. We overload the bool operator== as an example of an operator that returns a Boolean value. Other operators that returns Boolean values, such as >, >=, <, <=, != can be overloaded in a similar way.

18. Do copy/type in the example and see what happens when you run it to understand better how overloading friend operators work.

19. See also chapter 11 in the Study Guide, and Appendix 8 in Savitch for more information, as well as the solution to Assignment 2, specifically questions 6 and 7 for overloading >> and <<.

A program to illustrate operator overloading

The class below simulates a real number in a very restricted manner: the real number may have any number of digits before the full stop, but only one digit after the full stop. For example: 167.3 is a valid real number in our example, while 12.67 will not be valid

Our example overloads the following operators:

- `operator+` (to add two `RealNr`s)
- `operator++` (to increment a `RealNr` by 1)
- `operator==` (to compare two `RealNr`s)
- `operator<<` (to insert a `RealNr` in any output file – the console (screen) is also an output file)

- `operator>>` (to extract a `RealNr` from any input file – the keyboard is also an input file

The main program does the following (all indicated by comments in the main program):

- use mutators to assign values to two RealNr objects nr1 and nr2;
- use the overloaded operator<< to display values of nr1 and nr2 on the console,
- use the overloaded operator+ to add nr1 and nr and assign the result to nr3
- use accessors to display the value of nr3
- use overloaded operator >> to extract a value for nr4 from the keyboard
- use overloaded operator++ to increment nr4 by 1
- use overloaded operator<< to display the value of nr4 after it has been incremented
- use overloaded operator<< to display the value of nr1 and nr4
- compare the values of nr1 and nr4 using the overloaded bool operator==

RealNr.h:

```
#ifndef REALNR_H
#define REALNR_H
#include <iostream>

using namespace std;

class RealNr
{
    friend RealNr operator + (const RealNr& R1, const RealNr& R2);
    friend RealNr operator ++ (RealNr& R1);
    friend bool operator == (const RealNr& R1, const RealNr& R2);
    friend istream& operator >> (istream&, RealNr& r);
    friend ostream& operator << (ostream&, const RealNr& r);
    public:
        RealNr();
        ~RealNr();
        int getIntPart();
        int getDecPart();
        void setIntPart(int i);
        void setDecPart(int d);
    private:
    int intPart;
    int decPart;
};

#endif // REALNR_H
```

RealNr.cpp

```
#include "RealNr.h"

RealNr::RealNr(): intPart(0), decPart(0)
{

 }
```

```
RealNr::~RealNr()
{
    //dtor
}

  int RealNr::getIntPart()
 {
     return intPart;
 }

 int RealNr::getDecPart()
 {
     return decPart;
 }

 void RealNr::setIntPart(int i)
 {
     intPart = i;
 }

void RealNr::setDecPart(int d)

{
    decPart = d;
}

 istream& operator >> (istream& ins, RealNr& r)
 {
     char fullstop;
     ins >> r.intPart >> fullstop >> r.decPart;
     return ins;
 }

 ostream& operator << (ostream& outs, const RealNr& r)
 {
     outs << r.intPart << '.' << r.decPart;
     return outs;
 }

RealNr operator + (const RealNr& R1, const RealNr& R2)
{
    RealNr Temp;
    Temp.decPart  = R1.decPart + R2.decPart;
    Temp.intPart  = R1.intPart + R2.intPart;
    if (Temp.decPart > 10)
    {
        ++Temp.intPart;
        Temp.decPart -=10;
    }
    return Temp;
}

RealNr operator ++ (RealNr& R1)
{
   ++R1.intPart;
   return R1;
}

bool operator == (const RealNr& R1, const RealNr& R2)
    {
        if ((R1.intPart == R2.intPart)&& (R1.decPart == R2.decPart))
        return true;
        else return false;
    }
```

Main

```
#include <iostream>
#include "RealNr.h"

using namespace std;

int main()
{
    RealNr nr1, nr2, nr3, nr4;

//use mutators to assign values to nr1 and nr2
    nr1.setIntPart(1);
    nr1.setDecPart(5);
    nr2.setIntPart(5);
    nr2.setDecPart(9);

//use overloaded operator<< to display values of nr1 and nr2 on the
//console
    cout << "nr1 = " << nr1 << endl;
    cout << "nr2 = " << nr2 << endl;

//use overloaded operator+ to add nr1 and nr and assign the result
//to nr3
    nr3 = nr1 + nr2;

//use accessors to display the value of nr3
    cout << "After adding nr1 and nr2, nr3 is " << nr3.getIntPart()
         << '.' << nr3.getDecPart() << endl << endl;

//use overloaded operator >> to extract a value for nr4 from the
//keyboard
    cout << "Please enter a real nr in the following format: "
         <<"integerPart.decimalPart" << endl
        << "Note that the real number should have only one decimal "
        << "value " << endl
        << "in other words, only one digit after the decimal point "
        << endl
        <<  "Any number of digits is allowed before the decimal "
        << "point: ";
    cin >> nr4;
 1
//use overloaded operator++ to increment nr4 by
    ++nr4;
//use overloaded operator<< to display the value of nr4 after it has
//been incremented
    cout << endl << endl;
    cout << "After incrementing the real number, its value is "
         << nr4 ;

//use overloaded operator<< to display the value of nr1 and nr4
    cout << endl << endl;
    cout << "nr1 = " << nr1 << endl;
    cout << "nr4 = " << nr4 << endl;
    cout << endl << endl;

//compare the values of nr1 and nr4 using the overloaded bool
//operator==
    if (nr4 == nr1)
        cout << "nr4 is equal to nr1 " << endl;
    else cout << "nr4 is not equal to nr1 " << endl;

    return 0;
}
```

Output:

```
nr1 = 1.5
nr2 = 5.9
After adding nr1 and nr2, nr3 is 7.4

Please enter a real nr in the following format:
integerPart.decimalPart
Note that the real number should have only one decimal value
in other words, only one digit after the decimal point
Any number of digits is allowed before the decimal point: 0.5


After incrementing the real number, its value is 1.5

nr1 = 1.5
nr4 = 1.5


nr4 is equal to nr1

Process returned 0 (0x0)   execution time : 12.690 s
Press any key to continue.
```