In class Practice 10/25/2019

Group three or four people team.

Part I

1. Suppose a and b are integer variables and we form the sum a + b. Now suppose c and d are floating-point variables and we form the sum c + d. The two + operators here are clearly being used for different purposes. This is an example of _____operator overloeading_____.
2. Keyword _____operator_____introduces an overloaded-operator function definition.
3. To use operators on class objects, they must be overloaded, with the exception of operators_____ , _____and _____.assign(=),address(&),comma(,)
4. The operators that cannot be overloaded are _____, _____, _____and _____. “.”, “?:”, “::”, “.*”, “sizeof”, “typeid”
5. The ____new__ operator dynamically allocates memory for an object of a specified type and returns a(n) __pointer___ to that type.
6. The ___delete____ operator reclaims memory previously allocated by new.
7. Can we create new operator and overload it? no only existed operator can be overload
8. How does the precedence of an overloaded operator compare with the precedence of the original operator? the precedence is identical
9. (T/F)The following is a properly declared overloaded insertion operator for myClass. ostream& operator <<(ostream &out, const myClass &obj); true
10. Operators can be overloaded as D
    a. friends of a class
    b. members of a class
    c. non-friends, non-members of a class
    d. All of the above

Part II

Answer the following questions

a) What is the difference between new and new [], delete and delete []?
b) Write the function declaration for a destructor, a copy constructor, an assignment operator for a class named myClass.

new creats object and dynamically allocates space in memory for that object. delete destory a dynamically allocated object and free the space that was occupied by the object. new[] and delete[] have the similar function but they are applied to arrays.

```
~myClass();
myClass(const myClass& temp);
myClass operator=(const myClass& temp);
```

Part III

Q1 Create a Rational number (fractions) class. The following gives the definition of this class. The constructor set default value of 0 for numerator and 1 for denominator to prevent error. Function reduction divide both numerator and denominator by their gcd (greatest common divisor) if gcd is not 1. Implement addition (+) operator and multiplication (*) operators. In addition to these two arithmetic operators, implement == and != operators.

```
// RationalNumber class definition.
#ifndef RATIONAL_NUMBER_H
#define RATIONAL_NUMBER_H
#include <string>

class RationalNumber {
public:
   explicit RationalNumber(int = 0, int = 1); // default constructor
   RationalNumber operator+(const RationalNumber&); // addition
   RationalNumber operator-(const RationalNumber&); // subtraction
   RationalNumber operator*(const RationalNumber&); // multiplication
   RationalNumber operator/(const RationalNumber&); // division

   // relational operators
   bool operator<(const RationalNumber&) const;
   bool operator>(const RationalNumber&) const;
   bool operator<=(const RationalNumber&) const;
   bool operator>=(const RationalNumber&) const;

   // equality operators
   bool operator==(const RationalNumber&) const;
   bool operator!=(const RationalNumber&) const;

   std::string toString() const; // display rational number
private:
   int numerator; // private variable numerator
   int denominator; // private variable denominator
   void reduction(); // function for fraction reduction
};

#endif
```