Queens College, CUNY, Department of Computer Science
Object-Oriented Programming in C++
CSCI 211/611
Summer 2018
Instructor: Dr. Sateesh Mane

due date Monday, July 23, 2018, 11.59 pm

Homework: static data and methods

- **Experience with other classes has demonstrated that in many cases the source of difficulty is not the mathematics or the programming.**
- **The source of difficulty is the English (understanding the text).**
- **If you do not understand the words in the lectures or homework, THEN ASK.**
- **If you do not understand the concepts in the lectures or homework, THEN ASK.**
- **Send me an email, explain what you do not understand.**
- **Do not just keep quiet and then produce nonsense in exams.**

- **Consult your lab instructor for assistance.**
- **You may also contact me directly, but I cannot promise a prompt response.**
- Please submit your inquiry via email, as a file attachment, to `Sateesh.Mane@qc.cuny.edu`.
- Please submit one zip archive with all your files in it.
  1. The zip archive should have either of the names (CS211 or CS611):

     `StudentId_first_last_CS211_hw_static.zip`
     `StudentId_first_last_CS611_hw_static.zip`
  2. The archive should contain one "text file" named "hw_static.[txt/docx/pdf]" (if necessary) and cpp files named "Q1.cpp" and "Q2.cpp" etc.
  3. Note that not all questions may require a cpp file.
  4. A text file is not always required for every homework assignment.

General information

- You should include the following header files, to run the programs below.

```
#include <iostream>
#include <iomanip>
#include <string>
#include <cmath>
```

- If you require additional header files to do your work, feel free to include them.
- **Include the list of all header files you use, in your solution for each question.**
- The questions below do not require complicated mathematical calculations.
- If for any reason you require help with mathematical calculations, **ask the lab instructor or the lecturer.**

Q1 Class `StaticCAD`

- **Declare a class `StaticCAD` ("static, copy, assign, destroy") with static and non-static data.** All the data members are public, for simplicity.

```
class StaticCAD {
public:
  StaticCAD(int n) {
    ++num_obj;                                  // static
    len = n;
    pd = new double;                            // dynamic memory
    ps = new string[len];
  }

  // copy constructor
  StaticCAD(const StaticCAD &orig) {
    ++num_copy;                                 // static
    // write deep copy
  }

  // assignment operator
  StaticCAD& operator=(const StaticCAD &rhs) {
    if (this == &rhs) return *this;
    ++num_assign;                               // static
    // write deep copy

    return *this;
  }

  // destructor
  ~StaticCAD() {
    // release memory
  }

  static int num_obj, num_copy, num_assign;     // static
  int len;                                      // non-static
  double *pd;
  string *ps;
};

// static data initialization statements
int StaticCAD::num_obj = 0;
// etc
```

- **Initialize all the static data variables to zero.**

- **Write a deep copy in the copy constructor and assignment operator.**
  1. **Only the non-static data is copied** in the copy constructor and assignment operator.
  2. **The static data is not copied** in the copy constructor and assignment operator.
  3. Remember that static data belongs to the class, **not to individual objects.**
  4. The value of `num_obj` increments in the (non-default) constructor.
  5. The value of `num_copy` increments in the copy constructor.
  6. The value of `num_assign` increments in the assignment operator.
  7. It is just a simple way to count how many times the non-default constructor, copy constructor and assignment operator are invoked.
- **Release the dynamic memory correctly in the destructor.**
- Here is a simple main program.

```
#include <iostream>
#include <string>
using namespace std;

// class declaration

int main()
{
  StaticCAD c1(3), c2(4), c3(5);
  *c1.pd = 1.1;
  *c2.pd = 2.2;
  *c3.pd = 3.3;
  StaticCAD ccopy(c1);
  c3 = c1;
  c3 = c2;
  c3 = c3;

  cout << StaticCAD::num_obj << endl;          // class only, no object
  cout << StaticCAD::num_copy << endl;
  cout << StaticCAD::num_assign << endl;

  cout << endl;
  cout << c1.num_obj << endl;                 // we can also use objects
  cout << c2.num_obj << endl;                 // answer is the same
  cout << c2.num_copy << endl;
  cout << c3.num_copy << endl;
  cout << c1.num_assign << endl;
  cout << c3.num_assign << endl;
  return 0;
}
```

Q2 Class `Rational`

Q2.1 Data

- A **rational number** is one which can be expressed as a ratio of two integers, a numerator and a denominator.
- **Declare a class `Rational` with two private data members n and d of type `int`.**
- Obviously, n is the numerator and d is the denominator.
- Mathematically, there is a problem if the denominator equals zero, but we shall not worry about that here.

```
class Rational {
public:
  // etc

private:
  int n, d;
};
```

- We shall add some public and private methods to the class `Rational`.

Q2.2 Accessor methods

- **Write public const accessor methods to get the values of the numerator and denominator.**

```
int get_num() const      // fill in the rest
int get_den() const      // fill in the rest
```

Q2.3 Constructors

- **Write a default constructor to set $n = 1$ and $d = 1$.**
- **Write a non-default constructor with the following signature.**

```
Rational(int a, int b);
```

- *But do not set $n = a$ and $d = b$!*
- If a and b have a common factor, we want to cancel it.
- Mathematically, we cancel the **greatest common divisor (gcd)** of a and b.
- Hence write the following for the non-default constructor.

```
int c = gcd(a,b);          // greatest common divisor
n = a/c;
d = b/c;
```

- *The method "*`gcd`*" computes the greatest common divisor of a and b.*
- Our next step is therefore to write the method `gcd`.
- **Note:**
  1. The data members do not allocate memoey dynamically hence we do not need a deep copy.
  2. Therefore we do not need to write a copy constructor, assignment operator or destructor.
  3. The automatically generated versions by the compiler are adequate.

Q2.4 Private static method

- The method "`gcd`" computes the greatest common divisor of a and b.
- The algorithm to compute the greatest common divisor of two integers is given below.
- More important to us, **the algorithm is pure mathematics.**
  1. **Therefore "`gcd`" is a static class method.**
  2. It does not depend on the object.
  3. **The method `gcd` is also private.**
  4. It is for our own use only. We do not want external applications to mess with it.
- This is the algorithm.
  1. If $a < b$ return `gcd(b,a)`.
  2. Compute the remainder `c = a % b`.
  3. If $c == 0$ return b, else if $c == 1$ return 1, else return `gcd(b,c)`.
- We are also assuming that a and b are both positive, which is another weak point, but let us not spend time on too much math.
- **Just for practice, write the function non-inline.** The full code is given.

```
int Rational::gcd(int a, int b)
{
  if (a < b)
    return gcd(b,a);

  int c = a % b;
  if (c == 0)
    return b;
  else if (c == 1)
    return 1;

  return gcd(b,c);
}
```

- **Note the following.**
  1. **The keyword "`static`" only appears in the declaration.**
  2. The keyword "`static`" does not appear in the non-inline function body.
  3. When the function body is written non-inline, the syntax is the same as any other non-inline class method.

Q2.5 Operator + and − as class methods

- **Overload the operators + and − as public class methods.**
- They are both `const` because they do not change the "this object" of the operator.

```
Rational operator+(const Rational &r) const;
Rational operator-(const Rational &r) const;
```

- The sum of two rational numbers is given by the following formula.

$$\begin{aligned}\frac{n_1}{d_1} + \frac{n_2}{d_2} &= \frac{n_1 d_2 + n_2 d_1}{d_1 d_2} \\ n_{\text{sum}} &= n_1 d_2 + n_2 d_1 \\ d_{\text{sum}} &= d_1 d_2\end{aligned}$$

- The difference of two rational numbers is given by the following formula.

$$\begin{aligned}\frac{n_1}{d_1} - \frac{n_2}{d_2} &= \frac{n_1 d_2 - n_2 d_1}{d_1 d_2} \\ n_{\text{diff}} &= n_1 d_2 - n_2 d_1 \\ d_{\text{diff}} &= d_1 d_2\end{aligned}$$

- Hence your function bodies should look like this.

```
Rational operator+(const Rational &r) const {
  // compute nsum and dsum
  return Rational(nsum, dsum);
}

Rational operator-(const Rational &r) const {
  // compute ndiff and ddiff
  return Rational(ndiff, ddiff);
}
```

Q2.6 Class declaration

- Your overall class declaration should look like the following.

```
class Rational {
public:
  Rational();                                        // default constructor
  Rational(int a, int b);                            // non-default constructor

  int get_num() const;
  int get_den() const;

  Rational operator+(const Rational &r) const;       // public const operator
  Rational operator-(const Rational &r) const;       // public const operator

private:
  static int gcd(int a, int b);                      // private static method

  int n, d;
};

int Rational::gcd(int a, int b)                      // non-inline function body
{
  // etc
}
```

Q2.7 Main program

- Here is an example main program to test your code.

```
#include <iostream>
using namespace std;

// class declaration

int main()
{
  Rational r1(6,8), r2(2,4);

  cout << r1.get_num()  << "  " << r1.get_den()  << endl;
  cout << r2.get_num()  << "  " << r2.get_den()  << endl;

  Rational sum1 = r1 + r1;
  Rational sum2 = r1 + r2;
  Rational diff = r1 - r2;

  cout << sum1.get_num()  << "  " << sum1.get_den()  << endl;
  cout << sum2.get_num()  << "  " << sum2.get_den()  << endl;
  cout << diff.get_num()  << "  " << diff.get_den()  << endl;

  return 0;
}
```