10 Classes

REVIEW QUESTIONS

1. A class definition needs a semicolon at the end.

a. true

3. The name of a constructor must be different from the name of the class to which it belongs.

b. false

5. A programmer can change the value of *this* pointer.

b. false

7. A unary class function can have _____ instance(s) of the class.(s).

a. no

9. Public functions can be called by _____.

c. both functions inside and outside the class.

11. Private data in a class can accessed by _____.

d. both public and private functions in the class

13. A member in a *struct* is _____ by default.

b. public

EXERCISES

15. The semicolon is missing after the closing brace.

17. The destructor for the Fun class cannot specify any parameters.

19. The default constructor for the Fun class is declared as a private function, which would prevent the instantiation of any Fun objects.

21. Although there may be more than one private access specifier in a class declaration, it is recommended that there is only one of each type of access section in each class to make the documentation clearer.

23. The variable x is a private member. Since no access specifier is given, it will use the default access for a class, which is private.

25. The variable x is a private member. There is no access specifier for the area in which x is declared, therefore the default access, which is private, for a class will be used.

27. Class destructors may not have a return value, even *void*.

PROBLEMS

29.

```
/* ============== Fraction :: sub_from ============
   Subtract one Fraction from another
      Pre  fractions contain values
      Post difference stored in calling fraction
*/
void Fraction :: sub_from (const Fraction& fr2)
{
   numerator = (numerator     * fr2.denominator)
             - (fr2.numerator * denominator);
   denominator *= fr2.denominator;
   *this = Fraction (numerator, denominator);
   return;
}  // Fraction sub_from
```

31.

```
/* ============== Fraction :: div_by ================
   Divide one fraction by another
      Pre  fractions contain values
      Post quotient stored in calling fraction
*/
void Fraction :: div_by (const Fraction& fr2)
{
   numerator   *= fr2.denominator;
   denominator *= fr2.numerator;
   *this        = Fraction (numerator, denominator);
   return;
}  // Fraction div_by
```

33.

```
/* ================== mult_by ==================
   Multiply 2 fractions
      Pre  fractions contain values
      Post product returned
*/
Fraction mult_by (const Fraction& fr1,
                  const Fraction& fr2)
{
   int numen = fr1.numerator   * fr2.numerator;
   int denom = fr1.denominator * fr2.denominator;
   return Fraction (numen, denom);
}  // Fraction friend mult_by
```

35.

```
/* ================ equal_to ==============
   Determine if 2 fractions are equal
      Pre  fractions contain values
      Post returns true if equal
           returns false if not equal
*/
bool equal_to (const Fraction& fr1,
               const Fraction& fr2)
{
```

```
   bool result;
   if  (fr1.numerator * fr2.denominator
     == fr2.numerator * fr1.denominator)
           result = true;
   else
      result = false;
   return result;
}  // Fraction friend equal_to
```

37.

```
/*  ================= less_than =================
   Determine if 1 fraction is less than another
      Pre  fractions contain values
      Post returns true if fr1 < fr2
           returns false if fr1 >= fr2
*/
bool less_than (const Fraction& fr1,
                const Fraction& fr2)
{
   bool result;
   if  (fr1.numerator * fr2.denominator
      < fr2.numerator * fr1.denominator)
           result = true;
   else
      result = false;
   return result;
}  // Fraction friend less_than
```

39.

```
/* Revised Fraction class definition and test driver.
      Written by:
      Date:
*/
#include <iostream>
using namespace std;

class Fraction
{
   private:
      int numerator;
      int denominator;
      int  greatestComDiv (int n1, int n2);
   public:
           Fraction ( );
           Fraction (int numer);
           Fraction (int numer, int denom);
           Fraction (const Fraction& copyFrom);
          ~Fraction ()
             {cout <<
                 "In Fraction destructor\n";
             }
      void store    (int numer, int denom);
      void print    ( ) const;
};  // Fraction
/*  ==============  Fraction :: Fraction =========
   Constructor for Fraction class.
   Initializes fraction to zero.
       Pre   none
       Post  fraction object initialized to 0
*/
```

```
Fraction :: Fraction ()
{
   cout << "In Fraction default constructor\n";
   numerator   = 0;
   denominator = 1;
}  // constructor
/* ==============  Fraction :: Fraction ==========
   Default constructor for Fraction class
   Initializes fraction to values in parameter list.
       Pre   numen contains numerator value
       Post  fraction object initialized
*/
Fraction :: Fraction (int numen)
{
   cout << "In Fraction constructor (one argument)\n";
   numerator   = numen;
   denominator = 1;
}  // Fraction constructor
/* ==============  Fraction :: Fraction ==============
   Initializes fraction to values in parameter list
   ensuring that the fraction is normalized.
       Pre  numen and denom contain fraction values
       Post fraction object initialized
*/
Fraction :: Fraction (int numer, int denom)
{
   cout << "In Fraction constructor (two argments)\n";
   if (denom == 0)
      {
       cout << "Error: denominator is zero" << endl;
       exit (100);
      } // zero denom
   if (denom < 0)
      // Ensure that any negative is in numerator
      {
       denom = -denom;
       numer = -numer;
      } // demon < 0
   int gcd = greatestComDiv (abs(numer), abs(denom));
   numer = numer / gcd;
   denom = denom / gcd;

   numerator   = numer;
   denominator = denom;
}  // Constructor
/* ==================== copyFrom ====================
   Copy constructor for Fraction class.
       Pre   copyFrom exists and has values
       Post  new object created and data copied
*/
Fraction :: Fraction (const Fraction& copyFrom)
{
   cout << "In Fraction copy constructor\n";
   numerator   = copyFrom.numerator;
   denominator = copyFrom.denominator;
}  // Copy constructor
/* ================ Fraction :: store ==============
   Store the numerator and denominator in the fraction
   class. Calls constructor to ensure normalization.
       Pre   numer and denom contain the numerator
             and denominator respectively
```

```
        Post  data stored
*/
void Fraction :: store (int numer, int denom)
{
   *this = Fraction (numer, denom);
   return;
}  // Fraction store
/* ================ Fraction :: print ==============
   Prints the numerator and denominator as a fraction.
       Pre   fraction class must contain data
       Post  data printed
*/
void Fraction :: print () const
{
   cout << numerator << "/" << denominator;
   return;
}  // Fraction print
/* ================ greatestComDiv =================
   Determine the greatest common divisor of two
numbers.
      Pre  Given two integers
      Post GCD returned
*/
int Fraction:: greatestComDiv (int n1, int n2)
{
   if (n1 < n2)
      {
       int temp = n1;
       n1 = n2;
       n2 = temp;
      } // n1 < n2
   if (n2 == 0)
      return n1;
   else
      return greatestComDiv (n2, n1 % n2);
}  // greatestComDiv

// ================ End Class Functions ==============

int main ()
{
   cout << "=== Start Fraction test Program ===\n\n";
   Fraction fr1;
   Fraction fr2 (8, 32);
   Fraction fr3 (fr2);

   cout << "   fr1  =  ";
   fr1.print ();
   cout << "\n   fr2  =  ";
   fr2.print ();
   cout << "\n   fr3  =  ";
   fr3.print ();
   cout << endl;

   fr1  =  Fraction (12, 3);
   cout << "   fr1  =  ";
   fr1.print ();
   cout << "\n   fr2  =  ";
   fr2.print ();
   cout << "\n   fr3  =  ";
```

```
    cout << " === End of Fraction test Program ===\n\n";
    return 0;
}   // main
```