ET-580 – Object Relationships - Practice

1. Implement Composition:

   a. Class Number
      1. data members:
         n - an integer value
      2. one-parameter and default constructors
      3. accessor and mutator functions for n

   b. Class Rational Number
      1. data members
         num - numerator, Number type
         den - denominator, Number type
      2. two-parameter and default constructors
      3. overload << to print a Rational Number object

   c. Main
      1. Create the Rational Number 2/3 and print it to console.

Example Output

```
2/3
```

2. Implement Aggregation

   a. Class Professor
      1. data members:
            name - string value
      2. one-parameter and default constructors
      3. accessor and mutator functions for name
      4. overload << to print a Professor object

   b. Class Course
      1. data members
         num - course number, integer type
         prof - course instructor, Professor type
      2. Constructors:
            a. default constructor sets num to 0 and prof to nullptr
            b. constructor with int and string input parameters, sets num and creates a professor object using dynamic memory
            c. constructor with int and professor input parameters, sets num and points prof to the input professor object
      3. accessor and mutator for num
      4. accessor for prof which returns a pointer to the prof object
      5. mutator for prof which points prof to a new object
      6. Overload << to print a Course object

   c. Main
      1. Create a professor object p1
      2. Create a course object c1 using the professor object p1
      3. Create a course object c2 using a string for the professor name
      4. Use << to print c1 and c2
      5. Use << to print p1
      6. Use << and a course accessor to print c2.prof

Notes:

- The big three is not implemented because in aggregation the external object (prof in this scenario) is expected to manage its own memory
- The course constructor which accepts a professor string creates an object using dynamic memory. This object has an independent lifetime and therefore can function like an external entity.
- getProfessor could be coded to return a professor pointer or reference

<u>Example Output</u>

```
580 Trowbridge
580 An
Trowbridge
An
```

3. Implement Inheritance:

   a. Class *Person*
      1. data member:
         name – name of the person, automatic variable
      2. output function which prints name
      3. ensure that name is directly accessible by student

   b. Class *Student* derived from *Person*
      1. data member:
         id – eight-digit integer, automatic variable
      2. redefined output function which prints name and id

   c. *Main* Function:
      1. instantiate an automatic variable of type *Person* and type *Student*
      2. print both variables using the appropriate *output* function

   <u>Example Output</u>

```
Aragorn
Legolas 52345243
```

4. Clone and modify the previous program:

   a. Class *Person*
      1. data member: *name* – convert to a dynamic variable
      2. update member functions as needed
      3. implement the big three
         each big three function should print the function name (see example)

   b. Class *Student* derived from *Person*
      1. data member: *id* – convert to a dynamic variable
      2. update member functions as needed
      3. implement the big three
         each big three function should print the function name (see example)

   c. *Main* Function:
      1. create a student object s1 and print it
      2. test the copy constructor
      3. test the assignment overload operator

<u>Example Output</u>

```
Legolas 52345243

==> person: copy constructor
==> student: copy constructor
Legolas 52345243

==> student: assignment overload
==> person: assignment overload
Legolas 52345243
```