Instructions:

- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula/fact sheet. No calculators or other computing devices may be used.
- Answer each question in the space provided. If you need to continue an answer onto the back of a page, clearly indicate that and label the continuation with the question number.
- If you want partial credit, justify your answers, even when justification is not explicitly required.
- There are 7 questions, priced as marked. The maximum score is 100.
- When you have completed the test, sign the pledge at the bottom of this page and turn in the test.
- Note that failure to return this test, or to discuss its content with a student who has not taken it, is a violation of the Honor Code.

Do not start the test until instructed to do so!

Name  Solution

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.
1. [15 points] One of the code fragments shown below contains a memory leak and the other does not. Clearly identify where the memory leak occurs (not just which fragment) and then show a revision of the code that has the same logical effect but does not involve a memory leak.

```cpp
class DiceRoll {
private:
    int Sum;
public:
    DiceRoll(); // random throw
    int SumIs();
};

DiceRoll *thisRoll;
int Frequency[13] = {0};
for (int k=0; k<100; k++) {
    thisRoll = new DiceRoll;
    Frequency[thisRoll.SumIs()]++;
}
for (int j=2; j<=12; j++) {
    cout << setw(3) << j << setw(8) << Frequency[j] << endl;
}(A)
```

```cpp
class Complex {
private:
    double Real;
    double Imag;
public:
    Complex();
    double Modulus();
};
double TotalModulus = 0.0;
for (int k=0; k<100; k++) {
    Complex thisComplex;
    TotalModulus += thisComplex.Modulus();
}
cout << "Total: " << setw(8) << setprecision(4) << TotalModulus << endl;
(B)
```

The memory leak occurs within the first `for` loop in (A). Each time the loop body is entered, the invocation of `new` causes the allocation of storage for a new `DiceRoll` object as the target of the pointer `thisRoll`. Any previous target of `thisRoll` remains in memory, allocated to this process. The solution is to delete the previous target of `thisRoll` before creating a new one:

```cpp
for (int k=0; k<100; k++) {
    thisRoll = new DiceRoll;
    Frequency[thisRoll.SumIs()]++;
    delete thisRoll;
}
```

The corresponding for loop in (B) does not cause a memory leak because the allocation of a `Complex` object is not made on the heap, but is simply a local variable, created when it comes in scope.
2. [21 points] Consider the following code fragment:

```cpp
class PacMonster {
private:
    Location myLocation;
    Color    myColor;
    bool     gotMunchies;
public:
    PacMonster();                                         // # 1
    PacMonster(Location newLocation, Color newColor);     // # 2
    PacMonster(Location &newLocation);                    // # 3
    PacMonster(Location *newLocation);                    // # 4
    PacMonster(Color newColor, Location newLocation);     // # 5
    void setColor(Color newColor);                        // # 6
    void setLocation(Location newLocation);               // # 7
    void setLocation();                                   // # 8
    bool flipMunchies();                                  // # 9
    Color getColor() const;                               // #10
    Location getLocation() const;                         // #11
    bool hasMunchies() const;                              // #12
};
```

PacMonster Goober;
Location Center(200, 200);
Location *upperRight = new Location(400, 0);
Color    newColor = RED;

For each of the following, write the number corresponding to the member function that would be invoked; if there is no matching member function, write “none”.

(a) Goober.setLocation(Center);    #7

(b) Goober.setColor(YELLOW);    #6

(c) PacMonster Barney(upperRight);    #4

(d) PacMonster Barney(Center);    #3

(e) Goober.setLocation(newColor);    none

(f) PacMonster Floyd(*upperRight);    #3

(g) PacMonster Opie(Center, RED);    #2
3. [18 points] Write the roman numeral matching the definition of each term:

(a) Abstraction  ____ I ____
    I. a named, tangible representation of the attributes and behavior relevant to modeling a given entity for some particular purpose.

(b) Association  ____ VII ____
    II. a named software representation for an abstraction, which separates the implementation of the representation from the interface of the representation.

(c) Class  ____ II ____
    III. in object-oriented programming, the restriction of access to data within an object to only those methods defined by the object’s class.

(d) Object  ____ IV ____
    IV. a distinct instance of a given class that encapsulates its implementation details and is structurally identical to all other instances of that class.

(e) Composition  ____ V ____
    V. an organized collection of components interacting to achieve a coherent, common behavior.

(f) Encapsulation  ____ III ____
    VI. a composition that encapsulates the parts of the composition.

VII. a composition of independently constructed and externally visible parts.

VIII. an organization of abstractions into a directed graph, in which the arcs denote an “is-a” relation between a more generalized abstraction and the one or more specializations derived from it.

4. [15 points] Circle the letter corresponding to the best answer to each question:

(a) When an array of objects is declared …
   A. no constructor is invoked for the individual objects in the array.
   B. the programmer selects which constructor will be invoked for the individual objects in the array.
   C. the default constructor is automatically invoked for the individual objects in the array.
   D. None of these

(b) By default, an object parameter to a function is passed by …
   A. reference.
   B. value.
   C. pointer.
   D. None of the above, because objects cannot be passed as parameters.
   E. None of these

(c) Any variables defined in the public section of a class …
   A. are accessible only to the methods of only one instance of that class.
   B. are accessible only to the methods of every instance of that class.
   C. are hidden both conceptually and visually.
   D. are visible to aggregate classes that contain the class.
   E. None of these
(d) **Typically**, in a class definition …

A. the private section contains primarily data and the public section contains primarily access methods.
B. the public section contains primarily data and the private section contains primarily access methods.
C. the private section contains both data and access methods and the public section is empty.
D. the public section contains both data and access methods and the private section is empty.
E. None of these

(e) The organization of the code of a class into .h and .cpp files reflects the basic SE principle of …

A. generalization.
B. composition.
C. abstraction.
D. rationalization.
E. None of these

5. [15 points] Write the interface (declaration) of a class “Rational” that could be used as shown below. You need not show any part of the implementation of the member functions here. Use overloading where appropriate. Pass function parameters so as to achieve both efficiency and safety. Use const when appropriate.

```cpp
Rational Fraction1(17, 3); // represents the fraction 17/3
Rational Fraction2(1.25); // represents the fraction 125/100
Rational Fraction3; // represents the fraction 0 (0/1)
Fraction2.Reduce(); // reduces fraction to lowest terms (5/4)
Fraction1.isLessThan(Fraction2); // returns true if Fraction1 < Fraction2
Fraction1.isEqualTo(Fraction2); // returns true if Fraction1 == Fraction2
Fraction1.addTo(Fraction2); // returns sum of Fraction1 and Fraction2 as a Rational
```

```cpp
class Rational {
private:
    int Top, Bottom;

public:
    Rational();
    Rational(int aTop, int aBottom);
    Rational(double aDecimalValue);
    void Reduce();
    bool isLessThan(Rational Other) const;
    bool isEqualTo(Rational Other) const;
    Rational addTo(Rational Other) const;
};
```
6. [6 points] Show an efficient implementation for the member function isEqualTo() above. Perform only integer calculations.

```cpp
bool Rational::isEqualTo(Rational Other) const {
    return (Top * Other.Bottom == Bottom * Other.Top);
}
```

Recall that \((a/b) = (c/d)\) if and only if \(ad = bc\), assuming that \(b\) and \(d\) are nonzero.

7. [10 points] Consider the partial class declarations given below:

```cpp
class Foo {
private:
    Bar* BarListHead;
    Bar* currentBar;
    int NumBar;
    ...
public:
    Bar nextBar() const;
    void Advance();
    ...
};

class Bar {
private:
    double Length;
    double Diameter;
    bool Snaffled;
    ...
public:
    bool isSnaffled() const;
    double Mass() const;
    void flipSnaffled();
    ...
};

class Widget {
private:
    Bar myBar;
    ...
public:
    void Snaffle(Bar );
    void Grab(Bar& );
    ...
};
```

Which pair(s) of classes exhibit an acquaintance (has-a) relationship?
Foo contains two pointers to Bar objects; this could be viewed as acquaintance or aggregation.

Which pair(s) of classes exhibit an aggregation relationship?
Widget contains a data member of type Bar which would indicate an aggregation.