1 Short Answer

1. What is the difference between a protected class member and a private class member?

   Solution: protected members can be seen by derived classes whereas private members cannot.

2. Which constructor is called first, that of the derived class or the base class?

   Solution: The base class constructor is called first.

3. When does static binding take place? When does dynamic binding take place?

   Solution: Static binding takes place at compile time and dynamic binding takes place at run time.

4. What is an abstract base class?

   Solution: An abstract base class is one that contains at least one purely virtual function. These are defined with an = 0 at the end of their specification.

5. Write the first line of the declaration for a Poodle class. The class should be derived from the Dog class with public base class access.

   Solution: `class Poodle: public Dog`

6. Write the first line of the declaration for a SoundSystem class. Use multiple inheritance to base the class on the MP3player class, the Tuner class, and the CDPlayer class. Use public base class access in all cases.

   Solution: `class SoundSystem: public MP3player, public Tuner, public CDPlayer`

7. What is a container?

   Solution: A container is a class that stores data and organizes it in some fashion.

8. What is an iterator?

   Solution: An iterator is an object that behaves like a pointer. It is used to access the individual data elements in a container.

9. Write a divide function that takes in two parameters, numerator and denominator. If the denominator is 0 have the function throw an exception as a string containing "ERROR: Cannot divide by zero."

   Solution:

```
double divide(int numerator, int denominator) {
    if (denominator == 0) {
        string exceptionString = "ERROR: Cannot divide by zero.\n";
        throw exceptionString;
    }

    return static_cast<double>(numerator) / denominator;
}
```

10. Write the code for the main program that will call the above function and catch the exception if one is thrown. The block of code is to also print out the returned exception string.

    Solution:

```
try {
    quotient = divide(num1, num2);
    cout << "The quotient is " << quotient << endl;
} catch (string exceptionString) {
    cout << exceptionString;
}
```

2 Coding Exercise #1

This exercise is to write an inheritance structure for triangles. Recall from geometry that an Isosceles triangle is one with two equal sides and that an Equilateral triangle is one where all three sides are of equal length.

Write a Triangle class that has a default constructor and one that takes in the three sides as parameters. This class should store the lengths of the three sides of the triangle. In addition it should have the following functions.

- Area: this function will calculate and return the area of the triangle. Recall that if the sides of the triangle are a, b, and c then let p be the semi-perimeter $p = (a + b + c)/2$ then the area is

$$A = \sqrt{p(p-a)(p-b)(p-c)}$$

- Sides: this prints to the screen the lengths of the sides.
- Draw: this prints "Draw Triangle" to the screen.

Write an Isosceles class that inherits off the Triangle class. There must be a default constructor and one that takes two parameters, the first is the double sides of equal length and the third is the length of the last side. That is, `Isosceles(3, 5)` is a triangle with side lengths 3, 3, and 5. This class must also be able to call the functions Area, Sides, and Draw, but in this case the Draw command will print to the screen, "Draw Isosceles Triangle".

Write an Equilateral class that inherits off the Isosceles class. There must be a default constructor and one that takes one parameter, the length of all the sides. That is, `Equilateral(7)` is a triangle with side lengths 7, 7, and 7. This class must also be able to call the functions Area, Sides, and Draw, but in this case the Draw command will print to the screen, "Draw Equilateral Triangle".

Write the specifications and implementations for each of the three classes below. There is to be no inline code. There is a block of sample code below and its output. Read this very closely, your class structures are to produce exactly the same output to this sample code.

Sample Code

```
Triangle *tris[5];
tris[0] = new Triangle(3, 4, 5);
tris[1] = new Isosceles(3, 5);
tris[2] = new Isosceles(4, 5);
tris[3] = new Equilateral(7);
tris[4] = new Triangle(7, 5, 3);

for (int i = 0; i < 5; i++) {
    tris[i]->Draw();
    cout << tris[i]->Area() << endl;
    tris[i]->Sides();
    cout << endl;
}
```

Output

```
Draw Triangle
6
Side Lengths: 3 4 5

Draw Isosceles Triangle
4.14578
Side Lengths: 3 3 5

Draw Isosceles Triangle
7.80625
Side Lengths: 4 4 5

Draw Equilateral Triangle
21.2176
Side Lengths: 7 7 7

Draw Triangle
6.49519
Side Lengths: 7 5 3
```

Solution:

```
#ifndef TRIANGLE_H_
#define TRIANGLE_H_

class Triangle {
protected:
    double a;
    double b;
    double c;

public:
    Triangle(double sideA = 0, double
        sideB = 0, double sideC = 0);
    double Area();
    void Sides();
    virtual void Draw();
};

#endif /* TRIANGLE_H_ */
```

```
#include <cmath>
#include <iostream>

#include "Triangle.h"

using namespace std;

Triangle::Triangle(double sideA, double
    sideB, double sideC) {
    a = sideA;
    b = sideB;
    c = sideC;
}

double Triangle::Area() {
    double p = (a + b + c) / 2;
    double area = sqrt(p * (p - a) * (p -
        b) * (p - c));
    return area;
}

void Triangle::Sides() {
    cout << "Side Lengths: " << a << " "
        << b << " " << c << endl;
}

void Triangle::Draw() {
    cout << "Draw Triangle" << endl;
}
```

```
#ifndef ISOSCELES_H_
#define ISOSCELES_H_

#include "Triangle.h"

class Isosceles: public Triangle {
public:
    Isosceles(double d = 0, double t = 0);
    void Draw();
};

#endif /* ISOSCELES_H_ */
```

```
#include <iostream>

#include "Isosceles.h"

using namespace std;

Isosceles::Isosceles(double d, double t):
    Triangle(d, d, t) {
}

void Isosceles::Draw() {
    cout << "Draw Isosceles Triangle" <<
        endl;
}
```

```
#ifndef EQUILATERAL_H_
#define EQUILATERAL_H_

#include "Isosceles.h"

class Equilateral: public Isosceles {
public:
    Equilateral(double a = 0);
    void Draw();
};

#endif /* EQUILATERAL_H_ */
```

```
#include <iostream>

#include "Equilateral.h"

using namespace std;

Equilateral::Equilateral(double a):
    Isosceles(a, a) {
}

void Equilateral::Draw() {
    cout << "Draw Equilateral Triangle" <<
        endl;
}
```

3 Coding Exercise #2

This exercise is to write a complete templated class structure named Tarray. This structure will store an array of any type and have functions for access, finding the max and min, sorting, and printing of the array, as long as the structures being stored have streaming out and the two relational operators > and < defined.

Specifically, the Tarray class should store a dynamic array of any type and the size of the array. In addition, the class must implement the following.

- A constructor that brings in two parameters, the first is the size of the array and the second is the element that the array will be populated with. So for example, `Tarray<int> A(50, 0)` will create an integer array of size 50 and put 0 in each of the entries of the array.
- A destructor, obviously. Make sure that there are no memory leaks.
- `getSize` that returns the size of the array.
- `set` that takes as parameters a position and element. It puts the element in the specified position. If the position is invalid then the array is unaltered.
- `get` that takes as a parameter a position and returns the element in that position. This function does not need to do bounds checking.
- `Max` this returns the largest element in the array.
- `Min` this returns the smallest element in the array.
- `Sort` this sorts the array from smallest to largest.
- `Print` this prints the array to the screen on a single line with commas between the elements.
- Overload the `[]` operator, as with get you do not need to do bounds checking.

There is a block of sample code below and its output. Read this very closely, your class structure is to produce exactly the same output to this sample code. As usual, no inline code. The next three pages are for your answer.

Sample Block of Code

```
Tarray<int> intA(10, 4);
Tarray<string> strA(5, "Empty");
intA.Print();
strA.Print();

for (int i = 0; i < intA.getSize(); i++)
    intA.set((i * i + 7) % intA.getSize(), i);

strA.set(0, "Jack");
strA.set(1, "Jill");
strA.set(2, "Sam");
strA.set(3, "Tom");
strA.set(4, "Brenda");

intA.Print();
strA.Print();

cout << intA.get(6) << endl;
cout << strA.get(2) << endl;
cout << intA[2] << endl;
cout << strA[4] << endl;

cout << intA.Min() << " " << intA.Max() << endl;
cout << strA.Min() << " " << strA.Max() << endl;

intA.Sort();
strA.Sort();
intA.Print();
strA.Print();
```

Output

```
4, 4, 4, 4, 4, 4, 4, 4, 4, 4
Empty, Empty, Empty, Empty, Empty
4, 8, 5, 6, 4, 4, 7, 0, 9, 4
Jack, Jill, Sam, Tom, Brenda
7
Sam
5
Brenda
0 9
Brenda Tom
0, 4, 4, 4, 4, 5, 6, 7, 8, 9
Brenda, Jack, Jill, Sam, Tom
```

Solution:

```cpp
#ifndef TARRAY_H_
#define TARRAY_H_

#include <iostream>

using namespace std;

template<class T>
class Tarray {
private:
    T *arr;
    int size;

public:
    Tarray(int sz, T def);
    virtual ~Tarray();
    int getSize();

    void set(int, T);
    T get(int pos);
    T Max();
    T Min();

    T& operator[](int);
    void Sort();
    void Print();
};

template<class T>
int Tarray<T>::getSize(){
    return size;
}

template<class T>
Tarray<T>::Tarray(int sz, T def) {
    size = sz;
    arr = new T[size];
    for (int i = 0; i < size; i++)
        arr[i] = def;
}

template<class T>
Tarray<T>::~Tarray() {
    delete[] arr;
}

template<class T>
void Tarray<T>::set(int pos, T item) {
    if (pos < 0 || pos > size - 1)
        return;

    arr[pos] = item;
}

template<class T>
T Tarray<T>::get(int pos) {
    return arr[pos];
}

template<class T>
T Tarray<T>::Max() {
    T max = arr[0];
    for (int i = 0; i < size; i++)
        if (arr[i] > max)
            max = arr[i];

    return max;
}

template<class T>
T Tarray<T>::Min() {
    T min = arr[0];
    for (int i = 0; i < size; i++)
        if (arr[i] < min)
            min = arr[i];

    return min;
}

template<class T>
T& Tarray<T>::operator[](int pos) {
    return arr[pos];
}

template<class T>
void Tarray<T>::Sort() {
    for (int i = 0; i < size - 1; i++)
        for (int j = 0; j < size - 1 - i;
            j++)
            if (arr[j] > arr[j + 1]) {
                T temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
}

template<class T>
void Tarray<T>::Print() {
    for (int i = 0; i < size; i++)
        if (i < size - 1)
            cout << arr[i] << ", ";
        else
            cout << arr[i] << endl;
}

#endif /* TARRAY_H_ */
```