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. What does LIFO stand for and what data structure uses this type of access?

   Solution: LIFO — Last In First Out, the stack uses LIFO access.

10. Write a declaration for an STL stack that stores doubles and uses a vector as its underlying storage structure. In addition, write a segment of code that will push the numbers from 1 to 100 inclusively onto the stack so that when the stack is popped the numbers will be in ascending order. Finally, write a segment of code that will pop the stack and write out the contents to the console.

    Solution:

```cpp
#include <iostream>
#include <stack>
#include <vector>

using namespace std;

int main() {
    stack<double, vector<double>> s;

    for (int i = 100; i >= 1; i--)
        s.push(i);

    for (int i = 0; i < 100; i++) {
        cout << s.top() << " ";
        s.pop();
    }

    return 0;
}
```

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 linked list class with some basic functionality. The class is to be templated so that it can store any data type that supports assignment, streaming out, and equality testing (i.e. ==). Specifically, the class structure is to i,plement the following. As usual, there is not to be any inline code in the specification for any of the functions.

- The list node should be an internal private struct named `ListNode`.
- A default constructor only.
- A destructor, obviously. Make sure that there are no memory leaks.
- `appendNode` that takes a single parameter, the element to be added to the list, and appends the element on to the end of the list.
- `insertFront` that takes a single parameter, the element to be added to the list, and puts the element on to the front of the list.
- `deleteNode` that takes a single parameter, the element to be deleted from the list, and removes the first occurrence of the element from the list. If the element is not in the list then the list is unaltered.
- `displayList` that will display the list to the console screen horizontally.

There is a sample program below and its output. Read this very closely, your class structure is to produce exactly the same output to this sample code. The next three pages are for your answer.

```cpp
#include "LinkedList.h"
#include <iostream>

using namespace std;

int main() {
    LinkedList<int> list;

    list.appendNode(5);
    list.appendNode(2);
    list.appendNode(1);
    list.appendNode(3);
    list.appendNode(7);

    list.displayList();

    list.insertFront(10);
    list.insertFront(15);
    list.insertFront(25);

    list.displayList();

    list.deleteNode(2);
    list.displayList();
    list.deleteNode(15);
    list.displayList();
    list.deleteNode(12345);
    list.displayList();
    list.deleteNode(7);
    list.displayList();
    list.deleteNode(25);
    list.displayList();

    return 0;
}
```

Output

```
5 2 1 3 7
25 15 10 5 2 1 3 7
25 15 10 5 1 3 7
25 10 5 1 3 7
25 10 5 1 3 7
25 10 5 1 3
10 5 1 3
```

Solution:

```
#ifndef LINKEDLIST_H
#define LINKEDLIST_H

#include <iostream>

using namespace std;

template <class T> class LinkedList {
  private:
    struct ListNode {
        T value;
        ListNode *next;
    };

    ListNode *head;

  public:
    LinkedList();
    ~LinkedList();

    void appendNode(T);
    void insertFront(T);
    void deleteNode(T);
    void displayList() const;
};

template <class T> LinkedList<T>::
    LinkedList() { head = nullptr; }

template <class T> LinkedList<T>::~
    LinkedList() {
    ListNode *nodePtr;
    ListNode *nextNode;

    nodePtr = head;

    while (nodePtr != nullptr) {
        nextNode = nodePtr->next;
        delete nodePtr;
        nodePtr = nextNode;
    }
}

template <class T> void LinkedList<T>::
    appendNode(T newValue) {
    ListNode *newNode;
    ListNode *nodePtr;

    newNode = new ListNode;
    newNode->value = newValue;
    newNode->next = nullptr;

    if (!head)
        head = newNode;
    else {
        nodePtr = head;

        while (nodePtr->next)
            nodePtr = nodePtr->next;

        nodePtr->next = newNode;
    }
}

template <class T> void LinkedList<T>::
    insertFront(T newValue) {
    ListNode *newNode;
    newNode = new ListNode;
    newNode->value = newValue;
    newNode->next = nullptr;

    if (!head) {
        head = newNode;
    } else {
        newNode->next = head;
        head = newNode;
    }
}

template <class T> void LinkedList<T>::
    deleteNode(T searchValue) {
    ListNode *nodePtr;
    ListNode *previousNode;

    if (!head)
        return;

    if (head->value == searchValue) {
        nodePtr = head->next;
        delete head;
        head = nodePtr;
    } else {
        nodePtr = head;

        while (nodePtr != nullptr &&
            nodePtr->value != searchValue)
             {
            previousNode = nodePtr;
            nodePtr = nodePtr->next;
        }

        if (nodePtr) {
            previousNode->next = nodePtr->
                next;
            delete nodePtr;
        }
    }
}

template <class T> void LinkedList<T>::
    displayList() const {
    ListNode *nodePtr;
    nodePtr = head;

    while (nodePtr) {
        cout << nodePtr->value << " ";
        nodePtr = nodePtr->next;
    }
    cout << endl;
}

#endif
```