**Program 5**

CS 3450

“Filter Iterators”

In this program you will create a generic, *iterable* sequence class, and decorate the iterator with a *filtering* iterator. Here are the pertinent interfaces in C++ (adapt to your language as needed):

template<class T>

class Iterator {

public:

virtual void first() = 0;

virtual void next() = 0;

virtual bool isDone() const = 0;

virtual T current() const = 0;

virtual ~Iterator(){}

};

template<class T>

class Iterable {

public:

virtual Iterator<T>\* getIterator() = 0;

virtual ~Iterable(){}

};

template<class T>

class Sequence {

public:

virtual void add(const T&) = 0;

virtual int size() const = 0;

virtual int capacity() const = 0;

virtual T get(unsigned int) const = 0;

virtual ~Sequence(){}

};

template<class T>

class IterableSequence : public Sequence<T>, public Iterable<T> {

};

1. Create an iterable sequence named **MyArray**. To keep things simple, it will be a fixed-size sequence using an underlying array of **T**, whose size is determined by a constructor parameter.
2. Write an **Iterator** that iterates over **MyArray**. Of course, that’s what MyArray:: getIterator() returns.
3. Then, create a decorator for **Iterator** named **FilterIterator** that accepts an **Iterator** and a predicate function (or function object) at creation. A **FilterIterator** should only refer to items that satisfy its predicate.
4. Also create a decorator for **Iterator** named **ReverseIterator** that iterates backwards. The easiest way to do this is to use an internal stack. Think about the inherent problems with this.

Write a driver that shows your FilterIterator in action:

1. With two different types of elements
2. A FilterIterator that filters results from another FilterIterator
3. A FilterIterator that filters out everything
4. A ReverseIterator
5. A ReverseIterator that takes a FilterIterator, and thus outputs the results of the Filter in reverse order

Note: this is actually pretty easy, but be sure to handle cases where you filter out one or more elements right at the beginning, and one or elements at the end. Think carefully!

Here’s what the interface could look like in Java:

public interface Iterator<T>

{

public void first();

public void next();

public boolean isDone();

public T current();

}

public interface Iterable<T>

{

public Iterator<T> getIterator();

}

public interface Sequence<T>

{

public void add(T value);

public int size();

public int capacity();

public T get(int);

}

public interface IterableSequence<T> extends Iterable<T>, Sequence<T>

{

}

public class MyArray<T> implements IterableSequence<T>

{

// Fill in your stuff here

}

public class MyIterator<T> implements Iterator<T>

{

// Fill in your stuff here

}

public class MyFilterIterator<T> implements Iterator<T>

{

// Fill in your stuff here

}