1. Programming Practice – Markov Chain Algorithm and its C++ Implementation

C++ source code is available from the textbook's website:
http://cm.bell-labs.com/cm/cs/tpop/markov++.c

Prefix: A sequence of NPREF (usually NPREF=2) words occurring consecutively inside the given document.

Suffix: A single word occurring inside the given document right after a prefix.

For the sake of consistency of processing uniformly all words as suffixes and of marking the beginning and the end of the document we add NPREF non-existing words called NONWORD at the beginning of the document and a single NONWORD at the end.

Markov Algorithm
Set *x,y* to be the first two words in the text;
Loop:
    Randomly choose *z* one of the suffixes of the prefix *x,y*;
    Print *z*;
    Replace *x,y* by *y,z*;
    Repeat loop;

STL Data Structures

STL documentation**:**
A vector is a sequence that supports random access to elements, constant time insertion and removal of elements at the end, and linear time insertion and removal of elements at the beginning or in the middle. The number of elements in a vector may vary dynamically; memory management is automatic. Vector is the simplest of the STL container classes, and in many cases the most efficient.
Member functions: push_back, pop_back, size.

typedef deque<string> Prefix;

STL documentation:
A deque is very much like a vector. Like a vector, it is a sequence that supports random access to elements, constant time insertion and removal of elements at the end of the sequence, and linear time insertion and removal of elements in the middle. The main way in which deque differs from vector is that deque also supports constant time insertion and removal of elements at the beginning of the sequence.
Member functions: push_front, push_back, pop_front, pop_back.

map<Prefix, vector<string> > statetab;

STL documentation:
Map is a associative container that associates objects of type key with objects of type data. Map has the important property that inserting a new element into a map does not invalidate iterators that point to existing elements. Erasing an element from a map also does not invalidate any iterators, except, of course, for iterators that actually point to the element that is being erased.

The expression *statetab[prefix]* performs a lookup in *statetab* with *prefix* as a key and returns a reference to the desired entry; the vector is created if it does not exist already.

The main operation during build phase:
statetab[prefix].push_back(s);

The main operation during generate phase:
vector<string>& suf = statetab[prefix];
const string& w = suf[rand() % suf.size()];

Constants

```
const int  NPREF = 2;
const char NONWORD[] = "\n";
const int  MAXGEN = 10000; // maximum words generated
```

Functions

```
void    build(Prefix&, istream&);
void    generate(int nwords);
void    add(Prefix&, const string&);
```

The main method

```
int main(void)
{
        int     nwords = MAXGEN;
        Prefix prefix;  // current input prefix

        for (int i = 0; i < NPREF; i++)
                add(prefix, NONWORD);
        build(prefix, cin);
        add(prefix, NONWORD);
        generate(nwords);
        return 0;
}
```

The build function

```
// build: read input words, build state table
void build(Prefix& prefix, istream& in)
{
	string buf;

	while (in >> buf)
		add(prefix, buf);
}
```

The add function

```
// add: add word to suffix deque, update prefix
void add(Prefix& prefix, const string& s)
{
	if (prefix.size() == NPREF) {
		statetab[prefix].push_back(s);
		prefix.pop_front();
	}
	prefix.push_back(s);
}
```

The generate function

```
// generate: produce output, one word per line
void generate(int nwords)
{
	Prefix prefix;
	int i;

	for (i = 0; i < NPREF; i++)
		add(prefix, NONWORD);

	for (i = 0; i < nwords; i++) {
		vector<string>& suf = statetab[prefix];
		const string& w = suf[rand() % suf.size()];
		if (w == NONWORD)
			break;
		cout << w << "\n";
		prefix.pop_front();	// advance
		prefix.push_back(w);
	}
}
```

2. Java: Interfaces

Interfaces
Interfaces are sets of prototypes of methods without implementation.

Java Quick-Sort with Interfaces
The code is available in the directory java/quick_sort_interfaces.
See textbook p. 38 for explanations.

```
import java.io.*;

class QuickSortTest
{
	public static void main(String[] args)throws IOException
	{
		int N=5;

		//We place N integers in reverse order inside an Integer array A
		Integer[] A=new Integer[N];
		for(int i=0;i<N;i++)
			A[i]=new Integer(N-i);

		//We apply quicksort method for sorting integers
		quicksort(A,0,N-1,new Icmp());

		//We print out numbers after sorting
		System.out.println("Sorted numbers are:");
		for(int i=0;i<N;i++)
			System.out.println(A[i]);
		System.out.println();

		//We construct an input stream is
		//It allows us to read input text line by line
		InputStreamReader converter = new InputStreamReader(System.in);
		BufferedReader is = new BufferedReader(converter);

		//We prompt the user for N strings
		//We store input strings inside a String array B
		String[] B=new String[N];
		System.out.println("Enter input strings, one string per line:");
		for(int i=0;i<N;i++)
			B[i]=is.readLine();
		System.out.println();

		//We apply quicksort method for sorting strings
		quicksort(B,0,N-1,new Scmp());
```

```
		//We print out strings after sorting
		System.out.println("Sorted strings are:");
		for(int i=0;i<N;i++)
			System.out.println(B[i]);
	}

	public static void quicksort(Object[] v, int left, int right, Cmp cmp)
	{
		int i, last;
		if(left>=right)
			return;
		//move pivot to the v[left]
		swap(v, left, rand(left, right));
		last=left;
		//partition
		for(i=left+1; i<= right; i++)
			 if(cmp.cmp(v[i],v[left])<0)
				swap(v, ++last, i);
		//restore pivot element
		swap(v, left, last);
		//recursively sort each part
		quicksort(v, left, last-1, cmp);
		quicksort(v, last+1, right, cmp);
	}

	static void swap(Object[] v,int i,int j)
	{
		Object temp;

		temp=v[i];
		v[i]=v[j];
		v[j]=temp;
	}

	public static int rand(int left, int right)
	{
		return left+(int)(Math.random()*(right-left+1));
	}
}

interface Cmp
{
	int cmp(Object x, Object y);
}
```

```
class Icmp implements Cmp
{
	public int cmp(Object o1, Object o2)
	{
		int i1=((Integer) o1).intValue();
		int i2=((Integer) o2).intValue();
		if(i1<i2)
			return -1;
		else if(i1==i2)
			return 0;
		else
			return 1;
	}
}

class Scmp implements Cmp
{
	public int cmp(Object o1, Object o2)
	{
		String s1=(String) o1;
		String s2=(String) o2;
		return s1.compareTo(s2);
	}
}
```

Java Sort with Interfaces

The *sort* method of *Arrays* class sorts any array of objects, provided the objects are instances of a class implementing *Comparable* interface. The interface Comparable looks as follows:

```
public interface Comparable
{
	int compareTo(Object other);
}
```

In the example below we sort an array consisting of objects of Person class. The full code is contained in the directory java/person_sort_interfaces

```
import java.util.*;

public class PersonSortTest
{
	public static void main(String[] args)
	{
		Person[] P=new Person[3];

		P[0]=new Person("Harry", "Hacker", "Beatty_Rd", "Sprinfield", "NJ");
		P[0].setAge(22);
		P[0].setSex("male");
		P[1]=new Person("Carl", "Cracker", "Daylestown_Blvd", "Tredyffrin",
				"CA");
		P[1].setAge(21);
		P[1].setSex("male");
		P[2]=new Person("Tony", "Tester", "Gartt_Rd", "Upper_Darby", "NY");
		P[2].setAge(23);
		P[2].setSex("male");

		//We sort array P according to alphabetical ordering of last names
		Arrays.sort(P);

		//After sorting we print out first names and last names of objects of P
		for(int i=0;i<P.length;i++)
			System.out.println(P[i].getFirstName()+" "+P[i].getLastName());

	}
}

class Person implements Comparable
{
	// constructor
	public Person(String aFirst, String aLast, String aStreet, String aCity,
			String aState)
	{
		firstname=aFirst;
		lastname=aLast;
		street=aStreet;
		city=aCity;
		state=aState;
	}
```

```
        //methods
        public String getFirstName()
        {
                return firstname;
        }

        …

        public void setSex(String aSex)
        {
                sex=aSex;
        }

        //the implementation of compareTo method
        public int compareTo(Object otherObject)
        {
                Person other=(Person) otherObject;
                return lastname.compareTo(other.lastname);
        }

        //fields
        private String firstname;
        private String lastname;
        private String street;
        private String city;
        private String state;
        private String  sex="";
        private int age=0;

}
```