```
// ========================================================================
// OOMarkov2.cpp  Created by Eric Wasiolek  May 30, 2005
//
// This program inputs a DNA sequences and automatically calculates the Markov
// chain probability of that sequence according to a specified probability
// transition matrix.  This is the object-oriented version.
//
// Added:  The user can define their own transition probability matrix.
// The probability matrix is read from a file as is the DNA sequence.
// The program automatically checks to see if the transition probability matrix
// is valid:  it must have a) 16 values, b) all values must be between 0 and 1,
// and c) all rows of values (every 4 values) must sum to 1.0, otherwise the
// program prints and error message and exits.
//
// Added:  The program verifies that the DNA sequence is a valid sequence
// file, i.e., that it ONLY contains A,C,G, or T's, otherwise the program
// prints and error message and re-requests input.
// The next version will be GUI (Windows interface) based.
// ========================================================================

#include <iostream>
#include <string>
#include <iomanip>
#include <cmath>
#include <fstream>
#include <cctype>

using namespace std;

#define MULT_IDENT 1
#define NUM_NUCLEOTIDES 4
#define MAXLINES 100
#define MATRIX_FILE_ERROR 1
#define INIT_PROB_ERROR 1
```

```
#define ERROR_TERM .001

typedef enum Nucleotide{ A, C, G, T };
ifstream InFile;


class Markov {

// Declare private class variables
private:


        float tm[4][4];

        string Nucs;

        float InitProb;

// private methods, called by the public methods
private:
  Nucleotide char2enum(char);
  bool CheckNucleotides(string);
  ValidateTransProbMatrix(ifstream&);

// public methods, provide the user's interface to this class
public:
        string InputSeq();
        InputTransProbMatrix();
        float CalcTransProb(string);
        Markov();  // Constructor
        DefineMatrix(ifstream&);
        OutputTransProbMatrix();
        OutputSeqProb(float);
```

```
        SetInitProb(float);
        InputInitProb();

};
/*
int main()
{

        Markov DNASeq;

        DNASeq.InputTransProbMatrix();

        cout << endl << endl;

        DNASeq.InputInitProb();

        string seq = DNASeq.InputSeq();

        cout << endl << "The sequence input is:  " << seq;

        cout << endl << endl;

        DNASeq.OutputSeqProb(DNASeq.CalcTransProb(seq));


return 0;

}
*/

Markov::Markov()
{
```

```
        for(int i = 0; i < NUM_NUCLEOTIDES; ++i)
                for(int j=0; j <NUM_NUCLEOTIDES; ++j)
                        tm[i][j] = 0;

// Absolutely cannot define a static array in the constructor!!!!

        Nucs  = "ACGT";

        InitProb = MULT_IDENT;

}

Markov::DefineMatrix(ifstream& InputFile)
{
        if(!InputFile)
                cout << "Can't find the file! " << endl;
        else
        {
                ValidateTransProbMatrix(InputFile);
                InputFile.clear();  // Yes!  Must rewind file!
                InputFile.seekg(0);
                for(int i = 0; i < NUM_NUCLEOTIDES; ++i)
                        for(int j=0; j <NUM_NUCLEOTIDES; ++j)
                        {
                                InputFile >> tm[i][j];
                                cout << "Diagnostic:  " << tm[i][j] << endl;
                        }
        }
}

Markov::OutputTransProbMatrix()
{
```

```
	cout << "The transition probability matrix is: " << endl;
	cout << "\tA\tC\tG\tT" << endl;
	for(int i = 0; i < NUM_NUCLEOTIDES; ++i)
	{
		cout << Nucs[i], ":  ";
		for(int j = 0; j < NUM_NUCLEOTIDES; ++j)
			cout << fixed << setprecision(3) << "\t" << tm[i][j];
		cout << endl;
	}
}

Markov::OutputSeqProb(float path_prob)
{
	cout << "The Markov probability of this seq of nucleotides is:  ";
	cout << fixed << setprecision(10) << path_prob;
	cout << endl << endl;
	cout << "The Markov probability expressed as a log is:  ";
	cout << fixed << setprecision(2) << log10(path_prob);
	cout << endl << endl;
}

Nucleotide Markov::char2enum(char c)
{

	switch(c)
	{
	case 'A' : return A;
	case 'C' : return C;
	case 'G' : return G;
	case 'T' : return T;
	default    : cout << "Invalid nucleotide\n";
	}
```

```
}



float Markov::CalcTransProb(string seq)
{
        float transprob = InitProb;

        for (int k = 0; k < seq.length()-1; k++)
        {
                transprob *= tm[char2enum(seq[k])][char2enum(seq[k+1])];
        }

        return transprob;

}

string Markov::InputSeq()
{
        string Seq = "";
        string SubSeq[MAXLINES];
        char Source;
        string filename;
        int count = 0;

        cout << "Would you like to input a sequence from the keyboard (K) or ";
        cout << "from a File (F)? ";
        cin.get(Source);
        switch(toupper(Source))
        {
        case 'K' :  {
```

```
                    do{
                    cout << endl << endl << "What's the sequence? ";
                    cin >> Seq;
                    }
                    while(false == CheckNucleotides(Seq))
                    ;
                    break;
                }

        case 'F' :  {
                    cout << endl << "What's the file name? ";
                    cin >> filename;
                    InFile.open(filename.c_str());
                    if(!InFile)
                            {
                                    cout << endl << "Cannot find or open file!" << endl;
                                    cout << "Re-input the sequence or a valid filename!";
                                    cout << endl << endl;
                                    Seq = InputSeq();
                            }
                    else
                            {
                                    cout << endl << "File \"" << filename << "\" opened
successfully!" << endl;
                                    while(!InFile.eof())
                                            {
                                            InFile >> SubSeq[count];
                                       Seq += SubSeq[count];
                                            if(false == CheckNucleotides(Seq))
                                            {
                                                    cout << "Re-input the file with a
correct DNA Sequence";
                                                    cout << endl << "composed only of
A,C,G, or T! ";
```

```
                                                    cout << endl << endl;
                                                    Seq = InputSeq();
                                                }
                                            cout << "SubSeq = " << SubSeq[count] << endl;
                                            ++count;
                                                }
                                        }
                                    break;
                                }

        default :   {
                                    cout << endl << "You must input either a 'K' or an 'F' ";
                                    cout << endl << endl;
                                    Seq = InputSeq();
                                }
        }  // end switch

InFile.close();
return Seq;

} // end InputSeq() function

bool Markov::CheckNucleotides(string Sequence)
{
        int count = 0;
        for(int i = 0; i < Sequence.length(); ++i)
        {
                if(             Sequence[i] == 'A'
                   || Sequence[i] == 'C'
                            || Sequence[i] == 'G'
                            || Sequence[i] == 'T'
                 )
                ++count;
```

```
                ;
        }

                if(count == Sequence.length())
                        return true;
                else if (count < Sequence.length())
                {
                        cout << "The sequence may only contain 'A', 'C', 'G', or 'T' ";
                        cout << endl << "Re-input the sequence! " << endl << endl;
                        return false;
                }
}

Markov::ValidateTransProbMatrix(ifstream& MatrixFile)
{

        int count = 0;
        float sum = 0;
        bool problem = false;
        float value = 0;

        while(!MatrixFile.eof())
        {
                MatrixFile >> value;
                ++count;
                sum += value;
                if(!(value > 0 && value < 1))
                {
                        cout << "There is a transition probability is not between 0 and 1!" << endl << endl;
                        problem = true;
                }

                if(count % NUM_NUCLEOTIDES == 0)
```

```
                {
                        cout << "Sum at this point is:  " << fixed << setprecision(10) << sum << endl;
                        if((fabs(1-sum) >  ERROR_TERM) && (sum != 1))
                        {
                                cout << "Each row must sum to 1! Re-input transition probability ";
                           cout << "values for this row!" << endl << endl;
                                problem = true;
                        }

                        sum = 0;
                }
        }

        if((count - 1) != (NUM_NUCLEOTIDES*NUM_NUCLEOTIDES))
        {
                cout << "You must have exactly " << NUM_NUCLEOTIDES*NUM_NUCLEOTIDES << "
values ";
                cout << "in your transition probability matrix!" << endl << endl;
                problem = true;
        }

                if(problem == true)
                {
                        cout << "Your transition probability input matrix is not valid." << endl;
                        cout << endl << "Please fix it and re-run the program." << endl << endl;
                        cout << "Exiting!" << endl << endl;
                        exit(MATRIX_FILE_ERROR);
                }

}

Markov::InputTransProbMatrix()
{
```

```
	string FileName;
	cout << "What is the name of your Transition Probability Matrix File? ";
	cin >> FileName;

	ifstream TransMatrix;
	TransMatrix.open(FileName.c_str());

	// Just call these methods directly, they are part of the same class
	// as is this method, don't have to attach them to an object
	DefineMatrix(TransMatrix);
	OutputTransProbMatrix();

	cout << endl << endl;

}

Markov::SetInitProb(float InitialProb)
{
	if((InitialProb > 0) && (InitialProb < 1))
		InitProb = InitialProb;
	else
		{
		cout << "Initial probability must be between 0 and 1!";
		cout << endl << "Re-run program with a valid initial probability." << endl;
		cout << "Exiting!";
		exit(INIT_PROB_ERROR);
		}
}

Markov::InputInitProb()
{
	string Input;
	float InitialProb;
```

```
	float DefaultInitProb = 1;

	cout << "Would you like to define an initial probability?  (Y)es or (N)o :  ";

	cin >> Input;

	switch(toupper(Input[0]))
	{
		case 'Y' : {
					cout << "What is the initial probability? ";
					cin >> InitialProb;
					SetInitProb(InitialProb);
					cout << endl << endl;
					break;
				}

		case 'N' : {
					cout << "The initial probability will be set to the default: 1";
					cout << endl << endl;
					SetInitProb(DefaultInitProb);
					break;
				}

		default :  {
					cout << "Your answer must be either (Y)es or (N)o.";
					cout << endl << endl;
					InputInitProb();
					break;
				}
	}

}
```