```
#include "stdafx.h"
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>


#define N 5
#define M 32
#define T 150
#define utterrance 20
#define no_of_digits 10
#define test 10
#define p 12
#define frame_size 320

//all global variable declarations
static int qt_star1[T]={0};
static long double alpha1[T+1][N+1]={0};
static long double beta1[T+1][N+1]={0};
static int O1[utterrance+1][T+1]={0};
static long double A1[N+1][N+1]={0};
static long double B1[N+1][M+1]={0};
static int Pi1[N+1]={0,1,0,0,0,0};
static long double zhi1[T+1][N+1][N+1]={0};
static long double gamma1[T+1][N+1]={0};
static long double Delta1[86][6]={0};
static int Shi1[86][6]={0};
static long double avg_A1[N+1][N+1]={0};
static long double avg_B1[N+1][M+1]={0};

static int T_values1[13][utterrance+1]={0};


static double * calculate_RIs1(double *test_array,double *Ri)
{

        for(int i=0;i<=p;i++)
        {
                for(int j=1;j<=320-i;j++)
                {
                        Ri[i]+=test_array[j]*test_array[i+j];
                }
        }
        return Ri;
}

//calculating linear predictive co efficients
static double * calculate_ais1(double *test_array,double *R,double *a)
{
        double E[13]={0};
        double alpha11[13][13]={0};
        double K[13]={0};
        E[0]=R[0];
        for(int i=1;i<=12;i++)
        {
                double sum=0.0;
                for(int j=1;j<=i-1;j++)
                {
                        sum+=alpha1[i-1][j]*R[i-j];
                }
                K[i]=(R[i]-sum)/E[i-1];
                alpha1[i][i]=K[i];
                for(int j=1;j<=i-1;j++)
```

```
                {
                        alpha1[i][j]=alpha1[i-1][j]-K[i]*alpha1[i-1][i-j];
                }
                E[i]=(1-K[i]*K[i])*E[i-1];
        }
        for(int i=1;i<=12;i++)
        {
                a[i]=alpha1[12][i];
        }

        return a;
}
//calculating c values
static double * calculate_CIs1(double *a,double *R,double *c)
{

        //double c[13];
        double Sigma=R[0]*R[0];
        c[0]=log10(double (Sigma));
        for(int i=1;i<=p;i++)
        {
                double sum=0.0;
                for(int k=1;k<=i-1;k++)
                {
                        sum+=(double(k)/double(i))*c[k]*a[i-k];
                }
                c[i]=a[i]+sum;
        }

        return c;
}
//raised sine function
static void raised_sine_window1(double *w)
{
        for(int i=1;i<=12;i++)
                w[i]=1+6*sin(3.14159265*i/12);
}

static void resetting_values()
        {

                FILE *fp_a = fopen("ffA.txt","r");
                FILE *fp_b = fopen("ffB.txt","r");
                FILE *fp_pi = fopen("ffPi.txt","r");

                for(int i=1;i<=N;i++)
                        {
                                for(int j=1;j<=N;j++)
                                {

                                        fscanf(fp_a,"%lf",&A1[i][j]);
                                        //printf("%.10g ",A1[i][j]);
                                }
                                //printf("\n");
                        }
                for(int i=1;i<=N;i++)
                        {
                                for(int j=1;j<=M;j++)
                                {

                                        fscanf(fp_b,"%lf",&B1[i][j]);
                                        //printf("%.10g ",B1[i][j]);
                                }
                                //printf("\n");
                        }
```

```
                for(int i=1;i<=N;i++)
                        {
                                fscanf(fp_pi,"%d",&Pi1[i]);
                                //printf("%d",Pi1[i]);
                        }
                fclose(fp_a);
        fclose(fp_b);
        fclose(fp_pi);
}
static long double viterbi(int iterations,int iter)
        {
                        //-------------------initialisation-----------------------------------------
-------
                        for(int i=1;i<=N;i++)
                        {
                                Delta1[1][i]=Pi1[i]*B1[i][O1[iterations][1]];
                                Shi1[1][i]=0;
                        }
                        /*-----------------------recursion-----------------------------------------
-----*/
                        long double max_value=0;
                        int state_index=-1;
                        for(int t=2;t<=iter;t++)
                        {
                                for(int j=1;j<=N;j++)
                                {
                                        max_value=0;
                                        state_index=-1;
                                        for(int i=1;i<=N;i++)
                                        {
                                                double value=Delta1[t-1][i]*A1[i][j];
                                                if(max_value<=value)
                                                {
                                                        max_value=value;
                                                        state_index=i;
                                                }

                                        }
                                        Delta1[t][j]=max_value*B1[j][O1[iterations][t]];
                                        Shi1[t][j]=state_index;

                                }
                        }


                        long double p1=-1;
                        int qt1=0;
                        for(int i=1;i<=N;i++)
                        {
                                if(Delta1[iter][i]>p1)
                                {
                                        p1=Delta1[iter][i];
                                        qt1=i;
                                }
                        }
                        //---------------------------------termination-------------------------------
-----------------
                        qt_star1[iter]=qt1;
                        for(int t=iter-1;t>=1;t--)
                        {
                                qt_star1[t]=Shi1[t+1][qt_star1[t+1]];

                        }

                        return p1;
```

```
}
static long double forward_backward(int iteration,int iter)
{

        //---------------------------------------initiaisation-----------------------------------
-----------

        for(int i=1;i<=N;i++)
        {
                alpha1[1][i]=Pi1[i]*B1[i][O1[iteration][1]];

        }


        //-------------------------------------------induction-----------------------------------
--------------
        for(int t=1;t<=iter-1;t++)
        {
                for(int j=1;j<=N;j++)
                {
                        long double sum=0;
                        for(int i=1;i<=N;i++)
                        {
                                sum+=alpha1[t][i]*A1[i][j];
                        }
                        alpha1[t+1][j]=sum*B1[j][O1[iteration][t+1]];
                }
        }

        //-------------------------------------------termination---------------------------------
-------------------
        long double probability=0;
        for(int i=1;i<=N;i++)
                probability+=alpha1[iter][i];


        //---------------------------------Backward procedure--------------------------------------
-------------------
        //-----------------------------------initialisation-----------------------------------------
-------------------
        for(int i=1;i<=N;i++)
        {
                beta1[iter][i]=1;
        }

        //-------------------------------------induction-----------------------------------------
--------------------
        for(int t=iter-1;t>=1;t--)
        {
                for(int i=1;i<=N;i++)
                {
                        long double sum=0;
                        for(int j=1;j<=N;j++)
                        {
                                sum+=A1[i][j]*B1[j][O1[iteration][t+1]]*beta1[t+1][j];
                        }
                        beta1[t][i]=sum;
                }
        }

        return probability;
}
//function for implementing Bowm wech algorithm
static void bowm_welch(int iteration,long double probability,int iter)
```

```
{
        //---------------------------zhi matrix calculation-----------------------------------------
-------
        for(int t=1;t<=iter;t++)
        {
                for(int i=1;i<=N;i++)
                {
                        for(int j=1;j<=N;j++)
                        {
                                zhi1[t][i][j]=alpha1[t][i]*A1[i][j]*B1[j][O1[iteration]
[t+1]]*beta1[t+1][j];
                                zhi1[t][i][j]/=probability;
                        }
                }
        }
        //--------------------------------gamma matrix calculation---------------------------------
------------
        for(int t=1;t<=iter;t++)
        {
                for(int i=1;i<=N;i++)
                {
                        long double sum=0;
                        for(int j=1;j<=N;j++)
                        {
                                sum+=zhi1[t][i][j];
                        }
                        gamma1[t][i]=sum;
                }
        }
        for(int t=1;t<=iter;t++)
        {
                for(int i=1;i<=N;i++)
                {
                        long double sum=0;
                        for(int j=1;j<=N;j++)
                        {
                                sum=sum+alpha1[t][j]*beta1[t][j];
                        }
                        gamma1[t][i]=(alpha1[t][i]*beta1[t][i])/sum;
                }
        }


        for(int i=1;i<=N;i++)
        {
                Pi1[i]=gamma1[1][i];
        }

        //-------------------------------------------updated A matrix-------------------------------
------------------------------
        for(int i=1;i<=N;i++)
        {
                for(int j=1;j<=N;j++)
                {
                        long double zhi_sum=0;
                        long double gamma_sum=0;
                        for(int t=1;t<=iter-1;t++)
                        {
                                zhi_sum+=zhi1[t][i][j];
                                gamma_sum+=gamma1[t][i];
                        }
                        A1[i][j]=zhi_sum/gamma_sum;
                }
        }
```

```
	for(int i=1;i<=N;i++)
	{
		long double sum=0;
		long double max_val=0;
		int index=0;
		for(int j=1;j<=N;j++)
		{
			sum=sum+A1[i][j];
			if(max_val<A1[i][j])
				index=j;
		}
		long double diff=1-float(sum);
		if(diff!=0)
		A1[i][index]+=diff;

	}

	for(int j=1;j<=N;j++)
	{
		for(int k=1;k<=M;k++)
		{
			long double gamma_sum_k=0;
			long double gamma_simple=0;
			for(int t=1;t<=iter;t++)
			{
				gamma_simple+=gamma1[t][j];
				if(O1[iteration][t]==k)
					gamma_sum_k+=gamma1[t][j];
			}
			B1[j][k]=gamma_sum_k/gamma_simple;
		}
	}
	//----------------------------Updated B matrix-------------------------------------------
	for(int j=1;j<=N;j++)
	{
		for(int k=1;k<=M;k++)
		{
			if(B1[j][k]==0)
			{
				B1[j][k]+=1e-30;
			}
		}
	}
	for(int j=1;j<=N;j++)
	{
		long double sum=0;
		long double max_val=0;
		int index=-1;
		for(int k=1;k<=M;k++)
		{
			sum+=B1[j][k];
			if(max_val<B1[j][k])
			{
				max_val=B1[j][k];
				index=k;
			}

		}
		B1[j][index]-=(sum-1);
	}


}
static double calculate_maximum_amplitude1(FILE *fp)
```

```
{
        double pmax=0;
        double m=0.0;
        double nmax=0;
        double y=0.0;
        char a[100];
        int i=0;
        while(!feof(fp))
        {
                fgets(a,100,fp);
                y=_atoi64(a);
                if(y>0)
                {
                if(pmax<y)
                        pmax=y;
                }
                else
                {
                        if(nmax>y)
                                nmax=y;
                }

        }
        if(abs(nmax)>abs(pmax))
                m=nmax;
        else
                m=pmax;
        return abs(m);
}

static void generating_obs_seq_for_train(char * dir , char *word)
{
        printf("All the train utterrances are converting into observation sequence and are getting
stored in separate files!!!!\n");
        //-----------------------------for traning files-----------------------------------------
----------
        //T_values file holds the no. of observations in every observation sequences for traning data
        FILE *fp_t=fopen("T_values.txt","w");
        char filename[100];
        //observation_sequence_1 will have observation sequence for digit 1 (all 20 utterrance )
        //sprintf(filename,"%s\\observation_sequence_%s.txt",dir,word);
        //FILE *fp_obs_seq=fopen(filename,"w");
        //iterating over all the utterrances
        for(int utter=1;utter<=utterrance;utter++)
        {
                //fprintf(fp_obs_seq,"---------------------------%d----------------------\n",word);
                char file[100];
                sprintf(file,"%s\\%s_%d.txt",dir,word,utter);
                //taking one file at a time
                FILE *fp_word=fopen(file,"r");
                //calculating max_amplitide for normalisation
                float max_amp=calculate_maximum_amplitude1(fp_word);
                //rewinding the pointer
                rewind(fp_word);
                int c=0;
                //counting the no. of data in current file
                while(!feof(fp_word))
                {
                        double x=0;
                        fscanf(fp_word,"%lf",&x);
                        c++;
                }
                c=c-1;
                //rewinding the file pointer
                rewind(fp_word);
```

```
//word1 will store the entire file
long double *word1;
word1=(long double *)malloc( (c+1) * sizeof(long double));
//normalising the data before storing into the array
for(int i=1;i<=c;i++)
{
        long double x=0;
        fscanf(fp_word,"%lf",&x);
        word1[i]=x*(5000/max_amp);

}
int t=0;
//observation array will store one frame at a time
double observation[T+1][frame_size+1]={0};
int frame_shift=1;
//taking frames slided by 80 samples
for(int i=1;i<=T;i++)
{
        for(int j=1;j<=frame_size;j++)
        {
                observation[i][j]=word1[frame_shift];
                frame_shift++;
        }
        if(frame_shift-80+frame_size>c)
        {
                t=i;
                break;

        }
        else
                frame_shift-=80;
}

fprintf(fp_t,"%d  ",t);

double w[p+1]={0};
raised_sine_window1(w);
// calculating linear predictive co efficients
double Ri[T+1][p+1]={0};
for(int i=1;i<=t;i++)
{
        calculate_RIs1(observation[i],Ri[i]);
}


double Ai[T+1][p+1]={0};
for(int i=1;i<=t;i++)
{
        calculate_ais1(observation[i],Ri[i],Ai[i]);
}


//calculatng cepstral coefficients
double Ci[T+1][p+1]={0};
for(int i=1;i<=t;i++)
{
        calculate_CIs1(Ai[i],Ri[i],Ci[i]);
}
//applying raised sine window
for(int i=1;i<=t;i++)
{
        for(int j=1;j<=p;j++)
        {
                Ci[i][j]=Ci[i][j]*w[j];
        }
```

```
                }
                //opening codebook , codebook array will store the entire codebook
                FILE *fp_codebook=fopen("codebook11.txt","r");
                double codebook[M+1][p+1];
                for(int i=1;i<=M;i++)
                {
                        for(int j=1;j<=p;j++)
                        {
                                double y=0;
                                fscanf(fp_codebook,"%lf",&y);
                                codebook[i][j]=y;
                        }
                }
                //weights for calculating tokhura's distance
                double weight[13]={0,1.0,3.0,7.0,13.0,19.0,22.0,25.0,33.0,42.0,50.0,65.0,61.0};
                double dist=1000;
                //t is the no. of frames present in current file
                for(int frame=1;frame<=t;frame++)
                {
                        dist=1000;
                        int index=1;

                        for(int k=1;k<=M;k++)
                        {
                                long double dist1=0;
                                for(int j=1;j<=p;j++)
                                {
                                        dist1+=weight[j]*(Ci[frame][j]-codebook[k][j])*(Ci[frame][j]-
codebook[k][j]);

                                }
                                if(dist1<dist)
                                {
                                        index=k;
                                        dist=dist1;
                                }
                        }
                        //this will store the observation values
                        //fprintf(fp_obs_seq,"%d   ",index);
                }
                //fprintf(fp_obs_seq,"\n");
                fclose(fp_word);
                fclose(fp_codebook);
        }
        fprintf(fp_t,"\n");
        //fclose(fp_obs_seq);
        fclose(fp_t);
        //T_values is the array containing no. of frames in every utterrance
        FILE *tt=fopen("T_values.txt","r");
        for(int i=1;i<=1;i++)  //Considering only 1 observation sequence
        {
                for(int j=1;j<=20;j++)
                {
                        fscanf(tt,"%d",&T_values1[i][j]);
                }
        }
        fclose(tt);
        return ;
}

static void convergence_using_feedforward(int digit)
{
        for(int iteration=1;iteration<=utterrance;iteration++)
        {
```

```
                printf("For Observation sequence %d\n" ,iteration);
                        resetting_values();
                        //after getting values into the respective matrices
                        long double p_star=viterbi(iteration,T_values1[digit+1][iteration]);
                        //printf("%g \n",p_star);
                        //printf("%g ",prob);
                        long double probability=forward_backward(iteration,T_values1[digit+1]
[iteration]);
                        //printf("%g",probability);
                        bowm_welch(iteration,probability,T_values1[digit+1][iteration]);

                        long double prev_p_star=p_star;
                        p_star=viterbi(iteration,T_values1[digit+1][iteration]);
                        //printf("%g \n",p_star);
                        int no_of_iterations=2;
                        while(no_of_iterations!=200)
                        {
                                prev_p_star=p_star;
                                probability=forward_backward(iteration,T_values1[digit+1]
[iteration]);
                                bowm_welch(iteration,probability,T_values1[digit+1][iteration]);
                                p_star=viterbi(iteration,T_values1[digit+1][iteration]);
                                no_of_iterations++;

                        }
                        printf("P_star = %g \n",p_star);
                        printf("State sequence\n");
                        for(int i=1;i<=T_values1[digit+1][iteration];i++)
                        {
                                //printf("%d  ",O1[iteration][i]);
                                printf("%d  ",qt_star1[i]);
                        }
                        printf("\n");

                        printf("\nFinal converged A matrix----------------------------------------
---------------->\n");
                        for(int i=1;i<=N;i++)
                        {
                                for(int j=1;j<=N;j++)
                                {
                                        printf("%g  ",A1[i][j]);
                                }
                                printf("\n");
                        }
                        printf("Final converged B matrix----------------------------------------
------------->\n");
                        for(int i=1;i<=N;i++)
                        {
                                for(int j=1;j<=M;j++)
                                {
                                        printf("%g  ",B1[i][j]);
                                }
                                printf("\n");
                        }
                        for(int i=1;i<=N;i++)
                        {
                                for(int j=1;j<=N;j++)
                                {
                                        avg_A1[i][j]+=A1[i][j];
                                }
                        }
                        for(int i=1;i<=N;i++)
                        {
                                for(int j=1;j<=M;j++)
                                {
```

```
                                        avg_B1[i][j]+=B1[i][j];
                                }
                        }

printf("*****************************************************************\n");

        }

}

static void modelling(char *word , char *dir)
{
        int digit = 0;
        printf("*********************************Digit %d*****************************\n",digit);
        char file[100];
        generating_obs_seq_for_train(dir,word);
        sprintf(file,"%s\\observation_sequence_%s.txt",dir,word); //car\observation_sequence_car.txt
        //taking feed forward model
        FILE *fp_observation = fopen (file,"r");
        FILE *fp_a = fopen("ffA.txt","r");
        FILE *fp_b = fopen("ffB.txt","r");
        FILE *fp_pi = fopen("ffPi.txt","r");
        //making average matrix of A and B zero
        for(int i=1;i<=N;i++)
        {
                for(int j=1;j<=N;j++)
                {
                        avg_A1[i][j]=0;
                }
        }
        for(int i=1;i<=N;i++)
        {
                for(int j=1;j<=M;j++)
                {
                        avg_B1[i][j]=0;
                }
        }

        for(int i=1;i<=utterrance;i++)
        {
                for(int j=1;j<=T;j++)
                {
                        O1[i][j]=0;
                }
        }

        if(fp_observation == NULL)
                printf("File location not given properly");
        else
        {
                char a[100];
                int skip=0;
                while(!feof(fp_observation))
                {
                        int data=0;
                        for(int j=1;j<=utterrance;j++)
                        {

                                float z;
                                fgets(a,100,fp_observation);
                                skip++;
                                z=_atoi64(a);
                                if(skip>0)
                                {
                                        data++;
```

```
                                        for(int i=1;i<=T_values1[digit+1][data];i++)
                                        {
                                                fscanf(fp_observation,"%d",&O1[j][i]);
                                        }
                                        fgets(a,100,fp_observation);
                                }
                        }
                }
        }

        //calling convergence function for converging each model
        convergence_using_feedforward(digit);

        //finding average model
        printf("Average mode is as follows\n");
        char filename_A[100];
        char filename_B[100];
        sprintf(filename_A,"%s\\A_%s.txt",dir,word);
        sprintf(filename_B,"%s\\B_%s.txt",dir,word);
        FILE *f_A=fopen(filename_A,"w");
        FILE *f_B=fopen(filename_B,"w");

        for(int i=1;i<=N;i++)
        {
                for(int j=1;j<=N;j++)
                {
                        avg_A1[i][j]/=utterrance;
                }

        }
        printf("B matrix--------------------------->\n");
        for(int i=1;i<=N;i++)
        {
                for(int j=1;j<=M;j++)
                {
                        avg_B1[i][j]/=utterrance;
                }

        }
        for(int i=1;i<=N;i++)
        {
                for(int j=1;j<=N;j++)
                {
                        printf("%g ",avg_A1[i][j]);
                        fprintf(f_A,"%g    ",avg_A1[i][j]);
                }
                printf("\n");
                fprintf(f_A,"\n");

        }
        printf("B matrix--------------------------->\n");
        for(int i=1;i<=N;i++)
        {
                for(int j=1;j<=M;j++)
                {
                        printf("%g ",avg_B1[i][j]);
                        fprintf(f_B,"%g    ",avg_B1[i][j]);
                }
                printf("\n");
                fprintf(f_B,"\n");

        }

        fclose(f_A);
        fclose(f_B);
```

```
        fclose(fp_observation);
        fclose(fp_a);
        fclose(fp_b);
        fclose(fp_pi);
        return ;
}
```