```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define M 4  //ベクトル次元数
#define N 50 //サンプル数
#define NN 25//訓練サンプル数、又はテストサンプル数
#define O 3  //クラス数

#define P 0.333 //事前確率

/****プログラムの流れ****
1.変数や配列の宣言と初期化
2.プログラムにデータを渡す
3.訓練サンプルを用いて平均ベクトルの推定 (式2.39)
4.訓練サンプルを用いて共分散行列の推定(式2.40)
5.共分散行列の逆行列と行列式の計算
6.テストサンプルと各クラスとの距離を計算する(式2.43)
7.最も距離が近い(dの値が小さい)クラスへテストサンプルを識別する(式2.42)
8.識別されたテストサンプルが正しいクラスへ識別されているかをチェックする
9.全テストサンプルに対して識別を行い,誤識別されたテストサンプル数を数える
10.誤識別率を求める

クラス0=setosa,クラス1=versic,クラス2=virgin
*/

//1.0.変数,配列の宣言
        double data_setosa[M][N];//各ファイルのinputデータを格納する配列
        double data_versic[M][N];
        double data_virgin[M][N];

        double se_avector[M];   //各クラスの平均ベクトル
        double ve_avector[M];   //
        double vi_avector[M];   //

        double setosa_CM[M][M];  //共分散行列用配列
        double versic_CM[M][M];
        double virgin_CM[M][M];

        double setosa_CMI[M][M]; //共分散逆行列用配列
        double versic_CMI[M][M];
        double virgin_CMI[M][M];

        double training_setosa[M][NN];  //各クラスの訓練サンプルを格納する配列
        double training_versic[M][NN];
        double training_virgin[M][NN];

        double test_setosa[M][NN];      //各クラスのテストサンプルを格納する配列
        double test_versic[M][NN];
        double test_virgin[M][NN];

        double d_setosa[NN];            //テストサンプルと各クラスの距離用の配列
        double d_versic[NN];
        double d_virgin[NN];

        int judge[3];   //最小の距離を格納する配列
        int child;      //誤識別率計算時の分子

        double min;     //最小値
        int num_b[6]={0},num_c[6]={0};  //誤識別サンプル番号格納配列
                                        //num_c:versic用、num_b:virgin用
//ここまで


//1.1.配列初期化
double Initialization(){
        int j,k;
        for(k=0;k<M;k++){
                for(j=0;j<N;j++){
                        data_setosa[k][j]=0;data_versic[k][j]=0;data_virgin[k][j]=0;
                }
                        se_avector[k]=0;ve_avector[k]=0;vi_avector[k]=0;
        }

        for(k=0;k<M;k++){
                for(j=0;j<M;j++){
                        setosa_CM[k][j]=0;versic_CM[k][j]=0;virgin_CM[k][j]=0;
                        setosa_CMI[k][j]=0;versic_CMI[k][j]=0;virgin_CMI[k][j]=0;
                }
        }

        for(k=0;k<M;k++){
                for(j=0;j<NN;j++){
                        training_setosa[k][j]=0;training_versic[k][j]=0;training_virgin[k][j]=0;
                }
        }
```

```
        for(j=0;j<N;j++){
                d_virgin[j]=0;d_versic[j]=0;d_setosa[j]=0;
        }

        for(j=0;j<0;j++){
                judge[j]=0;

        }
        for(j=0;j<6;j++){
                num_b[j]=0;num_c[j]=0;
                min=0;
        }
}
//ここまで

//2.0プログラムにデータを渡す
void file_input(char *filename,double d[M][N]){ //ファイル名,input配列
        FILE *fp;
        int a=0,b=0;
        double temp=0.0;

        if((fp=fopen(filename,"r"))==NULL){
                printf("ファイル:%sがありません",filename);
                exit(1);
        }

        while(fscanf(fp,"%lf",&temp)!=EOF){
                d[a][b]=temp;
                if(b==N-1){a++;b=0;}
                else{b++;}
                temp=0;
        }
        fclose(fp);
}
//ここまで

//2.1訓練サンプル、テストサンプルをそれぞれ分割する
        /* クラス50個のサンプルのうち、前半25個を訓練サンプル、後半25個をテストサンプルとする */

void holdout(double d[M][N],double k[M][NN],double t[M][NN]){
        int i,j;
        for(i=0;i<M;i++){
                for(j=0;j<N;j++){
                        if(j < NN){                // jが25よりも小さい時、すなわち前半25個までを、
                                k[i][j]=d[i][j];  // 訓練サンプル用配列に格納する
                        }
                        else{
                                t[i][j-NN]=d[i][j];     // 後半25個を訓練サンプル用配列に格納
                        }
                }
        }
}

//ここまで

//3.訓練サンプルを用いて平均ベクトルの推定
void cal_vector(double d[M][NN],double av[M]){  //引数説明: 各クラスの訓練データ, 平均ベクトル
        int s,t;
        double sum_xi[M]={0.0};
        for(t=0;t<M;t++){
                for(s=0;s<NN;s++){
                        sum_xi[t]+=d[t][s];
                }
        }
        for(t=0;t<M;t++){
                av[t]=sum_xi[t]/NN;              //25個の訓練データで平均ベクトルを求める
        }
}
//ここまで

//4.共分散行列の推定
void cal_CovarianceM(double d[M][NN],double av[M],double output[M][M]){
 //引数説明: 各クラスのデータ, 平均ベクトル, 共分散出力配列
        int j,k,l;
        double sum[M][M]={0};

        for(l=0;l<M;l++){
                for(k=0;k<M;k++){
                        for(j=0;j<NN;j++){
                                sum[l][k]+=((d[l][j]-av[l])*(d[k][j]-av[l]));
                        }
                        output[l][k]=sum[l][k]/(NN-1);  //25個の訓練データで共分散行列を求める
                }
        }
}
```

```
//ここまで

//5.0.共分散行列の逆行列の計算
void cal_inverse(double a[M][M],double output[M][M]){//逆行列を求める行列,逆行列を格納する配列
        double temp=0;          //計算に用いる変数
        double d[M][M]={0};     //関数内で用いる行列用配列
        int i,j,k;              //for文カウンター

        for(j=0;j<M;j++){       //引数を関数内の変数に渡す
                for(i=0;i<M;i++){
                        d[j][i]=a[j][i];
                }
        }
        //単位行列の生成
        for(j=0;j<M;j++){
                for(i=0;i<M;i++)
                        if(i!=j) output[j][i]=0.0;
                        else     output[j][j]=1.0;
        }

        //掃き出し法
        for(i=0;i<M;i++){
                temp=1.0/d[i][i];
                for(j=0;j<M;j++){
                        d[i][j]*=temp;
                        output[i][j]*=temp;
                }
                for(j=0;j<M;j++){
                        if(i!=j){
                                temp=d[j][i];
                        for(k=0;k<M;k++){
                                d[j][k]=d[j][k]-(d[i][k]*temp);
                                output[j][k]=output[j][k]-(output[i][k]*temp);
                                }
                        }
                }
        }
}
 //ここまで

//5.1 行列式(式2.43の+の右側の計算)
double cal_Determinant(double a[M][M]){ //行列式を計算する共分散行列
        double det=1.0,buf=0.0;         //行列式格納変数,計算途中値の格納変数
        double ans=0.0;                 //計算結果格納
        double b[M][M]={0};             //関数内行列格納配列
        int n=M;
        int i,j,k;                      //for文カウンター

        for(j=0;j<M;j++){               //引数を関数内の変数に渡す
                for(i=0;i<M;i++){
                        b[j][i]=a[j][i];
                }
        }
        //三角行列を作成
        for(i=0;i<n;i++){
                for(j=0;j<n;j++){
                        if(i<j){
                                buf=b[i][j]/b[i][i];
                                        for(k=0;k<n;k++){
                                                b[k][j]-=(b[k][i]*buf);
                                        }
                                }
                        }
                }

        //対角部分の積
        for(i=0;i<n;i++){
                det*=b[i][i];
        }
        ans=pow(det,0.5);               //行列式の平方根を取る…①

        //return (log(ans/P));          //①を事前確率で割った値のlog値を返す

        return ( (log(ans) / 2) + log(3) );     //式 (2.42) を変形した数式でreturnする

}
//ここまで

//6.テストサンプルと各クラスとの距離を計算する
double cal_distance(double test[M],double av[M],double CM[M][M],double CMI[M][M],double d[NN],int a){
 //引数の説明：テストサンプル, 平均ベクトル, 共分散行列, 共分散逆行列, 距離を格納する配列, 配列番号を指定する配列
        double temp[M]={0};
        double temp1[M]={0};
        int j,k,l;
        for(j=0;j<M;j++){
                temp[j]=0.5*(test[j]-av[j]);            //テストサンプルから平均ベクトルを引いた値
```

```
        }
        for(k=0;k<M;k++){
                for(j=0;j<M;j++){
                        temp1[k]+=(temp[j]*CMI[j][k]);  //上記の値に共分散逆行列を掛ける
                }
        }

        for(j=0;j<M;j++){
                        d[a]+=(temp1[j]*temp[j]);
                }
        d[a]+=cal_Determinant(CM);              //5.1の値を距離に加算する
}
//ここまで

//メイン関数
int main(void){
        int a,b,i,k;    //カウント用変数
        double min=0;   //最小値を格納する変数
        int child=0;    //誤識別率計算時の分母
        int num_b[6]={0},num_c[6]={0};  //誤識別サンプル番号を格納する配列


        /*クラスsetosaの各テストサンプルについての誤識別*/

                //関数の働きや引数の説明については各関数の説明を参照

  //*************************************************************************************************//
        Initialization();                                       //初期化
        file_input("iris setosa.txt",data_setosa);              //各ファイルのインプット
        file_input("iris versicolor.txt",data_versic);
        file_input("iris virginica.txt",data_virgin);

        holdout(data_setosa,training_setosa,test_setosa);
        holdout(data_versic,training_versic,test_versic);
        holdout(data_virgin,training_virgin,test_virgin);

        cal_vector(training_setosa,se_avector);    //setosaの前半25個のデータを訓練サンプルとして平均ベクトルを求める
        cal_CovarianceM(training_setosa,se_avector,setosa_CM);  //共分散行列を求める
        cal_inverse(setosa_CM,setosa_CMI);                      //逆共分散行列を求める

        cal_vector(training_versic,ve_avector);    //versicolorの前半25個のデータを訓練サンプルとして平均ベクトルを求める
        cal_CovarianceM(training_versic,ve_avector,versic_CM);  //共分散行列を求める
        cal_inverse(versic_CM,versic_CMI);                      //逆共分散行列を求める

        cal_vector(training_virgin,vi_avector);    //verginicaの前半25個のデータを訓練サンプルとして平均ベクトルを求める
        cal_CovarianceM(training_virgin,vi_avector,virgin_CM);  //共分散行列を求める
        cal_inverse(virgin_CM,virgin_CMI);                      //逆共分散行列を求める

        for(k = 0; k < NN; k++){
                double test[M]={0};
                for(i=0;i<M;i++){
                        test[i]=test_setosa[i][k];              //k番目のデータをテストサンプルとする
                }

                // k番目のテストデータから、各クラスの距離を算出する
                cal_distance(test,se_avector,setosa_CM,setosa_CMI,d_setosa,k);
                cal_distance(test,ve_avector,versic_CM,versic_CMI,d_versic,k);
                cal_distance(test,vi_avector,virgin_CM,virgin_CMI,d_virgin,k);

        }


        for(b=0;b<NN;b++){              //最小の距離を判定する
                min=d_setosa[b];
                if(d_versic[b]<min){
                        min=d_versic[b];
                }
                if(d_virgin[b]<min){
                        min=d_virgin[b];
                }

                if(min==d_setosa[b]){
                        judge[0]++;
                }
                if(min==d_versic[b]){
                        judge[1]++;

                }
                if(min==d_virgin[b]){
                        judge[2]++;
                }
        }
        //結果出力
        printf("¥n----setosa----¥n");
        printf("setosaと判定された数:%d¥n",judge[0]);
        printf("versicと判定された数:%d¥n",judge[1]);
```

```
	printf("virginと判定された数:%d¥n",judge[2]);

	//*********************************************************************************************//
	//ここまで


	/*クラスversicolorの各テストサンプルについての誤識別*/

	//*********************************************************************************************//

	Initialization();//初期化
	file_input("iris setosa.txt",data_setosa);		//クラスsetosaのときと同じ計算手順
	file_input("iris versicolor.txt",data_versic);
	file_input("iris virginica.txt",data_virgin);

	holdout(data_setosa,training_setosa,test_setosa);
	holdout(data_versic,training_versic,test_versic);
	holdout(data_virgin,training_virgin,test_virgin);

	cal_vector(training_setosa,se_avector);	//setosaの前半25個のデータを訓練サンプルとして平均ベクトルを求める
	cal_CovarianceM(training_setosa,se_avector,setosa_CM);	//共分散行列,逆共分散行列を求める
	cal_inverse(setosa_CM,setosa_CMI);

	cal_vector(training_versic,ve_avector);	//versicolorの前半25個のデータを訓練サンプルとして平均ベクトルを求める
	cal_CovarianceM(training_versic,ve_avector,versic_CM);	//共分散行列,逆共分散行列を求める
	cal_inverse(versic_CM,versic_CMI);

	cal_vector(training_virgin,vi_avector);	//verginicaの前半25個のデータを訓練サンプルとして平均ベクトルを求める
	cal_CovarianceM(training_virgin,vi_avector,virgin_CM);	//共分散行列,逆共分散行列を求める
	cal_inverse(virgin_CM,virgin_CMI);

	for(k = 0; k < NN; k++){
		double test[M]={0};
		for(i=0;i<M;i++){
			test[i]=test_versic[i][k];		//k番目のデータをテストサンプルとする
		}

		cal_distance(test,se_avector,setosa_CM,setosa_CMI,d_setosa,k);	//各クラスの距離を算出する
		cal_distance(test,ve_avector,versic_CM,versic_CMI,d_versic,k);
		cal_distance(test,vi_avector,virgin_CM,virgin_CMI,d_virgin,k);

	}
	a=0;

	for(b=0;b<NN;b++){		//最小値判定
		min=d_setosa[b];
		if(d_versic[b]<min){
			min=d_versic[b];
		}
		if(d_virgin[b]<min){
			min=d_virgin[b];
		}

		if(min==d_setosa[b]){
			judge[0]++;
		}
		if(min==d_versic[b]){
			judge[1]++;
		}
		if(min==d_virgin[b]){
			judge[2]++;
			child++;//誤識別した数
			num_c[a]=b;
			a++;//配列用カウンタ
		}
	}
	//出力
	printf("¥n--versicolor--¥n");
	printf("setosaと判定された数:%d¥n",judge[0]);
	printf("versicと判定された数:%d¥n",judge[1]);
	printf("virginと判定された数:%d¥n",judge[2]);
	printf("¥nvirginicaと誤識別されたテストサンプル番号及びベクトル成分¥n");
	for(b=0;b<6;b++){
		if(num_c[b]==0){}
		else{
		printf("versicolor_number %d : ¥t",num_c[b]+1);  // サンプル番号及びベクトル成分の表示
		printf("%.1lf ¥t %.1lf ¥t %.1lf ¥t %.1lf¥n",test_versic[0][num_c[b]] ,test_versic[1][num_c[b]] ,
test_versic[2][num_c[b]] , test_versic[3][num_c[b]] );
		}
	}
		printf("¥n");


	//*********************************************************************************************//
	//ここまで
```

```
	/*クラスvirginicaの各テストサンプルについての誤識別*/

	//*****************************************************************************************//
	Initialization();
	file_input("iris setosa.txt",data_setosa);
	file_input("iris versicolor.txt",data_versic);
	file_input("iris virginica.txt",data_virgin);

	holdout(data_setosa,training_setosa,test_setosa);
	holdout(data_versic,training_versic,test_versic);
	holdout(data_virgin,training_virgin,test_virgin);

	cal_vector(training_setosa,se_avector);          //setosaの前半25個のデータを訓練サンプルとして平均ベクトル
	cal_CovarianceM(training_setosa,se_avector,setosa_CM);  //共分散行列,逆共分散行列を求める
	cal_inverse(setosa_CM,setosa_CMI);

	cal_vector(training_versic,ve_avector);          //versicolorの前半25個のデータを訓練サンプルとして平均ベクトル
	cal_CovarianceM(training_versic,ve_avector,versic_CM);  //共分散行列,逆共分散行列を求める
	cal_inverse(versic_CM,versic_CMI);

	cal_vector(training_virgin,vi_avector);          //verginicaの前半25個のデータを訓練サンプルとして平均ベクトル
	cal_CovarianceM(training_virgin,vi_avector,virgin_CM);  //共分散行列,逆共分散行列を求める
	cal_inverse(virgin_CM,virgin_CMI);

	for(k = 0; k < NN; k++){
		double test[M]={0};
		for(i=0;i<M;i++){
			test[i]=test_virgin[i][k];              //k番目のデータをテストサンプルとする
		}

		cal_distance(test,se_avector,setosa_CM,setosa_CMI,d_setosa,k);  //各クラスの距離を算出する
		cal_distance(test,ve_avector,versic_CM,versic_CMI,d_versic,k);
		cal_distance(test,vi_avector,virgin_CM,virgin_CMI,d_virgin,k);

	}


	for(b=0;b<NN;b++){              //最小値判定
		min=d_setosa[b];
		if(d_versic[b]<min){
			min=d_versic[b];
		}
		if(d_virgin[b]<min){
			min=d_virgin[b];
		}

		if(min==d_setosa[b]){
			judge[0]++;
		}
		if(min==d_versic[b]){
			judge[1]++;
			num_b[a]=b;
			child++;        //誤識別した数
			a++;            //配列カウント
		}
		if(min==d_virgin[b]){
			judge[2]++;
		}
	}

	//出力
	printf("¥n--virginica--¥n");
	printf("setosaと判定された数:%d¥n",judge[0]);
	printf("versicと判定された数:%d¥n",judge[1]);
	printf("virginと判定された数:%d¥n",judge[2]);
	printf("¥nversicolorと誤識別されたテストサンプル番号及びベクトル成分¥n");
	for(b=0;b<6;b++){
		if(num_b[b]==0){}
		else{
			printf("virginica_number %d : ¥t",num_b[b]+1);  // サンプル番号及びベクトル成分の表示
			printf("%.1lf ¥t %.1lf ¥t %.1lf ¥t %.1lf¥n",test_virgin[0][num_b[b]] ,test_virgin[1][num_b[b]] ,
test_virgin[2][num_b[b]] , test_virgin[3][num_b[b]] );
		}
	}
		printf("¥n");



	printf("¥n全テストサンプル%d個中%d個 誤識別¥n",NN*0,child );     //誤識別したサンプル数の出力
	printf("誤識別率:%.3lf%¥n",(double)child*100/(NN*0));           //誤識別率出力

//*****************************************************************************************//
//ここまで

}
```

```
//main関数終了


//*********************************************************//
//*********************************************************//
//*******************プ　ロ　グ　ラ　ム　終　了*********************//
//*********************************************************//
//*********************************************************//
```