บทที่ 7.

โปรแกรม

7.1 โปรแกรม VqKmeanCalculate

7.1.1 VQ_KmeanKDTree.h

```
#include "opencv2/opencv.hpp"
#include <vector>
#include <algorithm>

using namespace cv;

typedef struct{
	double B,G,R;
} RgbPixelDouble;

typedef struct{
	float B,G,R;
} RgbPixelFloat;

typedef struct{
	double C0 ,C1 ,C2;
} centerDouble;

typedef struct{
	float C0 ,C1 ,C2;
} centerFloat;

#pragma once
class VQ_KmeanKDTree
{
public:
	VQ_KmeanKDTree(void);
	~VQ_KmeanKDTree(void);
	void setDataImageBGR(Mat &srcImg);
	void setDataImageBGROpenCV(Mat &srcImg,bool isUnique);
	void kmeansExecute(int cluster,int loopNumber);
	void openCVKmeansExecute(int cluster,int loopNumber);
	bool kmeansExecuteOpenCV(int cluster_count,int loopNumber);
	bool writeDataYAML(char *filename);
	vector<int> eachClusterNumber;
	vector<centerDouble> eachDoubleClusterCenter;
	vector<centerFloat>  eachFloatClusterCenter;
private:
	vector<int> vectorPixelColorAll;
	vector<RgbPixelDouble> vectorDoubleUniquePixelRGB;
	vector<float>  vectorFloatUniquePixelRGB;
	Mat pointSamples;
	Mat centerCluster;
	Mat eachClusterCount;
	void uniqueData(void);
	void prepare2Double(void);
	void prepare2FloatForOpenCV(void);
	void prepareDataWithUnique(Mat &srcImg,Mat &pointSamples);
	void prepareDataWithOutUnique(Mat &srcImg,Mat &pointSamples);
};
```

7.1.2 VQ_KmeanKDTree.cpp

```
#include "VQ_KmeanKDTree.h"

VQ_KmeanKDTree::VQ_KmeanKDTree(void)
{
}

VQ_KmeanKDTree::~VQ_KmeanKDTree(void)
{
	if(this->pointSamples.refcount!=NULL)
		this->pointSamples.release();
}

void VQ_KmeanKDTree::setDataImageBGROpenCV(Mat &srcImg,bool isUnique)
{
	if (isUnique)
	{
		this->prepareDataWithUnique(srcImg,this->pointSamples);
	}else
	{
		this->prepareDataWithOutUnique(srcImg,this->pointSamples);
	}
}

void VQ_KmeanKDTree::prepareDataWithUnique(Mat &srcImg,Mat &pointSamples)
{
	int lenghtPixel = srcImg.rows * srcImg.cols;

	vector<int> vectorIntTemp(lenghtPixel);

	MatIterator_<Vec3b> it = srcImg.begin<Vec3b>(),it_end =
srcImg.end<Vec3b>();

	int tempRGBint;
	for(int i=0; it != it_end; ++it,++i)
	{
		tempRGBint = (int)(*it)[0];
		tempRGBint += ((int)(*it)[1])<<8;
		tempRGBint += ((int)(*it)[2])<<16;
		vectorIntTemp[i] = tempRGBint;
	}

	// unique process using STL
	sort( vectorIntTemp.begin(), vectorIntTemp.end() );
	vectorIntTemp.erase( unique( vectorIntTemp.begin(), vectorIntTemp.end() ),
vectorIntTemp.end() );

	Mat processMat(vectorIntTemp.size(),1, CV_MAKETYPE(CV_8U,3));

	it = processMat.begin<Vec3b>();
	it_end = processMat.end<Vec3b>();

	int intBGR;
	int mask = 255;
	for(int i=0; it != it_end; ++it,++i)
	{
		intBGR = vectorIntTemp[i];
		(*it)[0] =  saturate_cast<uchar>(intBGR & mask);
		(*it)[1] =  saturate_cast<uchar>((intBGR>>8) & mask);
		(*it)[2] =  saturate_cast<uchar>((intBGR>>16) & mask);
	}
	//Mat pointSamples;
```

```
	processMat.convertTo(pointSamples, CV_32FC3,1.0/255.0,0.0);
	processMat.release();

	//return pointSamples;
}

void VQ_KmeanKDTree::prepareDataWithOutUnique(Mat &srcImg,Mat &pointSamples)
{
	//Mat pointSamples;
	srcImg.convertTo(pointSamples, CV_32FC3,1.0/255.0,0.0);
	pointSamples = pointSamples.reshape(3, srcImg.rows*srcImg.cols);
	//return pointSamples;
}

void VQ_KmeanKDTree::setDataImageBGR(Mat &srcImg)
{
	int lenghtPixel = srcImg.rows * srcImg.cols;
	this->vectorPixelColorAll.resize(lenghtPixel);
	uchar* data     = (uchar *) srcImg.data;
	int intBRG;
	for (int i =0;i<lenghtPixel;i++)
	{
		intBRG = (int)(*(data++));
		intBRG += (int)(*(data++))<<8;
		intBRG += (int)(*(data++))<<16;
		this->vectorPixelColorAll[i] = intBRG;
	}
	this->uniqueData();
}

void VQ_KmeanKDTree::uniqueData(void)
{
	sort( vectorPixelColorAll.begin(), vectorPixelColorAll.end() );
	vectorPixelColorAll.erase( unique( vectorPixelColorAll.begin(),
vectorPixelColorAll.end() ), vectorPixelColorAll.end() );
}
void VQ_KmeanKDTree::prepare2Double(void)
{
	int nPts = vectorPixelColorAll.size();
	this->vectorDoubleUniquePixelRGB.resize(nPts);
	int intBGR;
	int mask = 255;
	for (int j = 0; j < nPts; j++) {
		intBGR = vectorPixelColorAll[j];
		this->vectorDoubleUniquePixelRGB[j].B =  ((double)(intBGR &
mask))/255.0;
		this->vectorDoubleUniquePixelRGB[j].G =  ((double)((intBGR>>8) &
mask))/255.0;
		this->vectorDoubleUniquePixelRGB[j].R =  ((double)((intBGR>>16) &
mask))/255.0;
	}
}

void VQ_KmeanKDTree::prepare2FloatForOpenCV(void)
{
	int nPts = vectorPixelColorAll.size();
	this->vectorFloatUniquePixelRGB.resize((nPts*3));

	int intBGR;
	int mask = 255;
	for (int j = 0; j < nPts; j++) {
		intBGR = vectorPixelColorAll[j];
		this->vectorFloatUniquePixelRGB[(j*3)] =  ((float)(intBGR &
mask))/255.0f;
```

```
            this->vectorFloatUniquePixelRGB[(j*3+1)] =  ((float)((intBGR>>8) &
mask))/255.0f;
            this->vectorFloatUniquePixelRGB[(j*3+2)] =  ((float)((intBGR>>16) &
mask))/255.0f;
      }
}

void VQ_KmeanKDTree::openCVKmeansExecute(int cluster,int loopNumber)
{
      int cluster_count = cluster; /* number of cluster */
      this->prepare2FloatForOpenCV();

      Mat src_img = imread("c:"\baboon.jpg");

      int lenghtPixel = src_img.rows * src_img.cols;

      vector<int> vectorIntTemp(lenghtPixel);

      MatIterator_<Vec3b> it = src_img.begin<Vec3b>(),
            it_end = src_img.end<Vec3b>();
      int tempRGBint;

      for(int i=0; it != it_end; ++it,++i)
      {
            tempRGBint = (int)(*it)[0];
            tempRGBint += ((int)(*it)[1])<<8;
            tempRGBint += ((int)(*it)[2])<<16;
            vectorIntTemp[i] = tempRGBint;
      }

      cout << "non unique number :" << vectorIntTemp.size() << endl;
      sort( vectorIntTemp.begin(), vectorIntTemp.end() );
      vectorIntTemp.erase( unique( vectorIntTemp.begin(), vectorIntTemp.end() ),
vectorIntTemp.end() );
      cout << "unique number :" << vectorIntTemp.size() << endl;

      Mat processMat(vectorIntTemp.size(),1, CV_MAKETYPE(CV_8U,3));

      it = processMat.begin<Vec3b>();
      it_end = processMat.end<Vec3b>();

      int intBGR;
      int mask = 255;
      for(int i=0; it != it_end; ++it,++i)
      {
            intBGR = vectorIntTemp[i];
            (*it)[0] =  saturate_cast<uchar>(intBGR & mask);
            (*it)[1] =  saturate_cast<uchar>((intBGR>>8) & mask);
            (*it)[2] =  saturate_cast<uchar>((intBGR>>16) & mask);
      }

      Mat pointSamples;
      processMat.convertTo(pointSamples, CV_32FC3,1.0/255.0,0.0);
      Mat_<int> clusters(pointSamples.size(), CV_32SC1);
      // (3)run k-means clustering algorithm to segment pixels in RGB color
space
      //Mat_<int> clusters(points.size(), CV_32SC1);
      //cv::Mat centers(cluster_count,1,CV_32FC3);
      cv::Mat centers;
      //kmeans(pointSamples, cluster_count,
clusters,cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, loopNumber, 1.0), 1,
KMEANS_PP_CENTERS, &centers);
```

```
	kmeans(pointSamples, cluster_count,
clusters,cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, loopNumber, 1.0), 1,
KMEANS_PP_CENTERS, &centers);
	centers = centers.reshape(3, cluster_count);

	MatIterator_<Vec3f> itf     = centers.begin<Vec3f>();
	MatIterator_<Vec3f> itf_end = centers.end<Vec3f>();

	for(int i=0; itf != itf_end; ++itf,++i) {
		cout << (*itf)[0] << " ";
		cout << (*itf)[1] << " ";
		cout << (*itf)[2]<< endl;
	}
}

bool VQ_KmeanKDTree::kmeansExecuteOpenCV(int cluster_count,int loopNumber)
{
	if ((this->pointSamples.refcount==NULL))
		return false;

	Mat clusters = cv::Mat_<int>(this->pointSamples.size(),CV_32SC1);

	// (3)run k-means clustering algorithm to segment pixels in RGB color
space
	//kmeans(pointSamples, cluster_count,
clusters,cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, loopNumber, 1.0), 1,
KMEANS_PP_CENTERS, &centers);
	//kmeans(pointSamples, cluster_count,
clusters,cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, loopNumber, 1.0), 1,
KMEANS_USE_INITIAL_LABELS, &centers);
	cv::kmeans(pointSamples, cluster_count,
clusters,cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, loopNumber, 0.5), 1,
KMEANS_PP_CENTERS, &this->centerCluster);

	this->centerCluster = this->centerCluster.reshape(3, cluster_count);
	MatIterator_<Vec3f> itf     = this->centerCluster.begin<Vec3f>();
	MatIterator_<Vec3f> itf_end = this->centerCluster.end<Vec3f>();

	this->eachClusterCount = cv::Mat_<float>(cluster_count,1,CV_32FC1);
	this->eachClusterCount.setTo(cv::Scalar(0.0));
	MatIterator_<Vec<float, 1>> itff         = this-
>eachClusterCount.begin<Vec<float, 1>>();

	Mat tempCount = cv::Mat_<int>(cluster_count,1,CV_32SC1);
	tempCount.setTo(cv::Scalar(0.0));

	MatIterator_<Vec<int, 1>> itI         = tempCount.begin<Vec<int, 1>>();
	MatIterator_<Vec<int, 1>> itI_end     = tempCount.end  <Vec<int, 1>>();
	MatIterator_<Vec<int, 1>> itItmp;

	MatIterator_<Vec<int, 1>> itIC     = clusters.begin<Vec<int, 1>>();
	MatIterator_<Vec<int, 1>> itIC_end = clusters.end<Vec<int, 1>>();

	for(int i = 0; itIC != itIC_end; ++itIC) {
		int temp = (*itIC)[0];
		itItmp = itI+temp;
		(*itItmp)[0]++;
	}

	float totalPixel  = (float)this->pointSamples.rows;
	for(int i = 0; itI != itI_end; ++itI,++itff,++i) {
		(*itff)[0] = ((float)((*itI)[0]))/totalPixel;
		cout <<i<<" :: "<< (*itff)[0] << " ";
		cout << endl;
```

```
	}
	return true;
}

bool VQ_KmeanKDTree::writeDataYAML(char *filename)
{
	if (this->centerCluster.refcount==NULL)
		return false;
	cv::FileStorage fs(filename, cv::FileStorage::WRITE);
	fs << "ClusterCenter" << this->centerCluster;
	fs << "ClusterWeight" << this->eachClusterCount;
	fs.release();
	return true;
}

void VQ_KmeanKDTree::kmeansExecute(int clusterNumber,int loopNumber)
{
	KMterm  term(loopNumber, 0, 0, 0,            // run for 100 stages
		0.10, 0.10, 3,             // other typical parameter values
		0.50, 10, 0.95);

	this->prepare2Double();
	int dim = 3;
	int nPts = this->vectorDoubleUniquePixelRGB.size();
	KMpointArray source = kmAllocPts(nPts, dim);
	for (int j =0;j<nPts;j++)
	{
		source[j][0] = this->vectorDoubleUniquePixelRGB[j].B;
		source[j][1] = this->vectorDoubleUniquePixelRGB[j].G;
		source[j][2] = this->vectorDoubleUniquePixelRGB[j].R;
	}
	KMdata *dataPts = new KMdata(dim, nPts);
	kmCopyPts(nPts,dim,source,dataPts->getPts());

	kmDeallocPts(source);
	dataPts->buildKcTree();              // build filtering structure

	KMfilterCenters *ctrs = new KMfilterCenters(clusterNumber, *dataPts);
	// allocate centers
	KMlocalEZ_Hybrid kmAlg(*ctrs, term);
	*ctrs = kmAlg.execute();                    // execute

	double*           sqDist = new double[nPts];
	KMctrIdxArray newCands = new KMctrIdx[nPts];
	ctrs->getAssignments(newCands,sqDist);
	eachClusterNumber.resize(clusterNumber);

	for (int j =0;j<nPts;j++)
	{
		eachClusterNumber[newCands[j]]++;
	}
	delete []sqDist;
	delete []newCands;

	eachDoubleClusterCenter.resize(clusterNumber);
	KMcenterArray centerPoint = ctrs->getCtrPts();
	for (int j =0;j<clusterNumber;j++)
	{
		eachDoubleClusterCenter[j].C0 = centerPoint[j][0];
		eachDoubleClusterCenter[j].C1 = centerPoint[j][1];
		eachDoubleClusterCenter[j].C2 = centerPoint[j][2];
	}
	delete ctrs;
	delete dataPts;
```

```
	for (int j =0;j<clusterNumber;j++)
	{
		cout << eachDoubleClusterCenter[j].C0 << " , ";
		cout << eachDoubleClusterCenter[j].C1 << " , ";
		cout << eachDoubleClusterCenter[j].C2 << endl;
	}
	return;
}
```

7.1.3 kmeanCluster.h

```
#include "opencv2/opencv.hpp"
#include <vector>
#include <algorithm>

using namespace cv;

#pragma once
class kmeanCluster
{
public:
	kmeanCluster(void);
	~kmeanCluster(void);
	void setDataImageBGROpenCV(Mat &srcImg,bool isUnique);
	bool kmeansExecuteOpenCV(int cluster_count,int loopNumber);
	bool writeDataYAML(const char *filename);

private:
	Mat pointSamples;
	Mat centerCluster;
	Mat eachClusterCount;
	void prepareDataWithUnique(Mat &srcImg,Mat &pointSamples);
	void prepareDataWithOutUnique(Mat &srcImg,Mat &pointSamples);
};
```

7.1.4 kmeanCluster.cpp

```
#include "kmeanCluster.h"

kmeanCluster::kmeanCluster(void)
{
}

kmeanCluster::~kmeanCluster(void)
{
}

void kmeanCluster::setDataImageBGROpenCV(Mat &srcImg,bool isUnique)
{
	if (isUnique)
	{
		this->prepareDataWithUnique(srcImg,this->pointSamples);
	}else
	{
		this->prepareDataWithOutUnique(srcImg,this->pointSamples);
	}
}

void kmeanCluster::prepareDataWithUnique(Mat &srcImg,Mat &pointSamples)
{
	int lenghtPixel = srcImg.rows * srcImg.cols;
```

```
	vector<int> vectorIntTemp(lenghtPixel);

	MatIterator_<Vec3b> it = srcImg.begin<Vec3b>(),it_end =
srcImg.end<Vec3b>();

	int tempRGBint;
	for(int i=0; it != it_end; ++it,++i)
	{
		tempRGBint = (int)(*it)[0];
		tempRGBint += ((int)(*it)[1])<<8;
		tempRGBint += ((int)(*it)[2])<<16;
		vectorIntTemp[i] = tempRGBint;
	}

	// unique process using STL
	sort( vectorIntTemp.begin(), vectorIntTemp.end() );
	vectorIntTemp.erase( unique( vectorIntTemp.begin(), vectorIntTemp.end() ),
vectorIntTemp.end() );

	Mat processMat(vectorIntTemp.size(),1, CV_MAKETYPE(CV_8U,3));

	it = processMat.begin<Vec3b>();
	it_end = processMat.end<Vec3b>();

	int intBGR;
	int mask = 255;
	for(int i=0; it != it_end; ++it,++i)
	{
		intBGR = vectorIntTemp[i];
		(*it)[0] =  saturate_cast<uchar>(intBGR & mask);
		(*it)[1] =  saturate_cast<uchar>((intBGR>>8) & mask);
		(*it)[2] =  saturate_cast<uchar>((intBGR>>16) & mask);
	}
	//Mat pointSamples;
	processMat.convertTo(pointSamples, CV_32FC3,1.0/255.0,0.0);
	processMat.release();
	//return pointSamples;
}

void kmeanCluster::prepareDataWithOutUnique(Mat &srcImg,Mat &pointSamples)
{
	srcImg.convertTo(pointSamples, CV_32FC3,1.0/255.0,0.0);
	pointSamples = pointSamples.reshape(3, srcImg.rows*srcImg.cols);
}

bool kmeanCluster::kmeansExecuteOpenCV(int cluster_count,int loopNumber)
{
	if ((this->pointSamples.refcount==NULL))
		return false;

	Mat clusters = cv::Mat_<int>(this->pointSamples.size(),CV_32SC1);

	// (3)run k-means clustering algorithm to segment pixels in RGB color
space
	//kmeans(pointSamples, cluster_count,
clusters,cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, loopNumber, 1.0), 1,
KMEANS_PP_CENTERS, &centers);
	//kmeans(pointSamples, cluster_count,
clusters,cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, loopNumber, 1.0), 1,
KMEANS_USE_INITIAL_LABELS, &centers);
	cv::kmeans(pointSamples, cluster_count,
clusters,cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, loopNumber, 0.5), 1,
KMEANS_PP_CENTERS, this->centerCluster);
	this->centerCluster = this->centerCluster.reshape(3, cluster_count);
```

```
	MatIterator_<Vec3f> itf     = this->centerCluster.begin<Vec3f>();
	MatIterator_<Vec3f> itf_end = this->centerCluster.end<Vec3f>();

	this->eachClusterCount = cv::Mat_<float>(cluster_count,1,CV_32FC1);
	this->eachClusterCount.setTo(cv::Scalar(0.0));
	MatIterator_<Vec<float, 1>> itff         = this-
>eachClusterCount.begin<Vec<float, 1>>();

	Mat tempCount = cv::Mat_<int>(cluster_count,1,CV_32SC1);
	tempCount.setTo(cv::Scalar(0.0));

	MatIterator_<Vec<int, 1>> itI         = tempCount.begin<Vec<int, 1>>();
	MatIterator_<Vec<int, 1>> itI_end     = tempCount.end  <Vec<int, 1>>();
	MatIterator_<Vec<int, 1>> itItmp;

	MatIterator_<Vec<int, 1>> itIC     = clusters.begin<Vec<int, 1>>();
	MatIterator_<Vec<int, 1>> itIC_end = clusters.end<Vec<int, 1>>();

	for(int i = 0; itIC != itIC_end; ++itIC) {
		int temp = (*itIC)[0];
		itItmp = itI+temp;
		(*itItmp)[0]++;
	}

	float totalPixel  = (float)this->pointSamples.rows;
	for(int i = 0; itI != itI_end; ++itI,++itff,++i) {
		(*itff)[0] = ((float)((*itI)[0]))/totalPixel;
	}
	return true;
}

bool kmeanCluster::writeDataYAML(const char *filename)
{
	if (this->centerCluster.refcount==NULL)
		return false;
	cv::FileStorage fs(filename, cv::FileStorage::WRITE);
	fs << "ClusterCenter" << this->centerCluster;
	fs << "ClusterWeight" << this->eachClusterCount;
	fs.release();
	return true;
}
```

7.2 โปรแกรม VqRetrieval

7.2.1 yamlWriteRead.h

```
#include "opencv2/opencv.hpp"
#include <vector>
#include <algorithm>
using namespace cv;

#pragma once
class yamlWriteRead
{
public:
	bool readDataVq(char* filename, Mat &centerCluster, Mat
&eachClusterCount);
	bool readDataVqConst(const char* filename, Mat &centerCluster, Mat
&eachClusterCount);
	bool writeDataYAML(char* filename, Mat &centerCluster, Mat
&eachClusterCount);
	yamlWriteRead(void);
```

```
        ~yamlWriteRead(void);
};
```

7.2.2 yamlWriteRead.cpp

```
#include "yamlWriteRead.h"

yamlWriteRead::yamlWriteRead(void)
{
}

yamlWriteRead::~yamlWriteRead(void)
{
}

bool yamlWriteRead::readDataVq(char* filename, Mat &centerCluster, Mat
&eachClusterCount)
{
        FileStorage fsRead(filename, FileStorage::READ);

        fsRead["ClusterCenter"] >> centerCluster;
        fsRead["ClusterWeight"] >> eachClusterCount;
        fsRead.release();
        return true;
}

bool yamlWriteRead::readDataVqConst(const char* filename, Mat &centerCluster,
Mat &eachClusterCount)
{
        FileStorage fsRead(filename, FileStorage::READ);

        fsRead["ClusterCenter"] >> centerCluster;
        fsRead["ClusterWeight"] >> eachClusterCount;
        fsRead.release();
        return true;
}

bool yamlWriteRead::writeDataYAML(char* filename, Mat &centerCluster, Mat
&eachClusterCount)
{
        cv::FileStorage fs(filename, cv::FileStorage::WRITE);
        fs << "ClusterCenter" << centerCluster;
        fs << "ClusterWeight" << eachClusterCount;
        fs.release();
        return true;
}
```

7.2.3 VqRetrieval.cpp

```
#include "yamlWriteRead.h"
#include <opencv2/opencv.hpp>
#include <emmintrin.h>
#include <vector>
#include <string>
#include <libmysqlwrapped.h>
#include <mysql.h>

#include <iostream>                    // for std::cout
#include <utility>                     // for std::pair
#include <algorithm>                   // for std::for_each
#include <boost/graph/graph_traits.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/dijkstra_shortest_paths.hpp>
```

```
#include <boost/config.hpp>
#include <iostream>
#include <boost/graph/prim_minimum_spanning_tree.hpp>

#include <vector>
#include <math.h>

#include "kmeanCluster.h"
using namespace boost;

using namespace cv;
using namespace std;

struct Centet3Dfloat {
	float c0;
	float c1;
	float c2;
} ;

struct dbInfo{
	long Id;
	string imagePath;
	string yamlPath;
};

struct dbConnect{
	string dbHost;
	string dbUser;
	string dbPass;
	string dbDBName;
};

vector<dbInfo> readDataDbMysql(dbConnect dbConn,string DBID);
void MWWTestCompute(Mat & sampleMat,Mat & compareMat,Mat &samplePopulaWeight,Mat
& comparePopulaWeight,int &MWWTestVaule,double & MWWTestWeightVaule);
void MWWTestWeightCompute(void);

int main(int argc, char **argv)
{
	// 0 name program
	// 1 filepath image input
	// 2 cluster number
	// 3 loop number
	// 4 db file for sqlite
	// 5 flag for 0 for MWWTest other for MWWTestWeight

	if (argc != 10)
		return -1;

	const char *imagename = argv[1];
	const int clusterNum  = atoi(argv[2]);
	const int loopNum     = atoi(argv[3]);
	const int flagChooseMWW = atoi(argv[4]);

	dbConnect dbConn;
	string DBID;
	dbConn.dbHost   = argv[5];
	dbConn.dbUser   = argv[6];
	dbConn.dbPass   = argv[7];

	if (dbConn.dbPass.compare("null")==0 || dbConn.dbPass.compare("NULL")==0)
	{
		dbConn.dbPass = "";
```

```
	}

	dbConn.dbDBName = argv[8];
	DBID            = argv[9];

	if (DBID.compare("null")==0 || DBID.compare("NULL")==0)
	{
		DBID = "";
	}

	if (!(clusterNum < 100 && clusterNum > 1 && loopNum < 200 && loopNum >
10))
		return -1;

	Mat src_img = imread(imagename, 1);
	if(!src_img.data || src_img.channels()!=3)
		return -1;

	kmeanCluster kCluster;
	kCluster.setDataImageBGROpenCV(src_img,false);

	if (!(kCluster.kmeansExecuteOpenCV(clusterNum,loopNum)))
		return -1;

	Mat sampleCenterCluster;
	Mat sampleEachClusterCount;
	kCluster.getKmeanData(sampleCenterCluster,sampleEachClusterCount);
	yamlWriteRead yamlWR;

	vector<dbInfo> vecListFile = readDataDbMysql(dbConn,DBID);

	if (vecListFile.size()<1)
	{
		return -1;
	}

	vector<std::pair<Mat,Mat>> vqDataAllImage(vecListFile.size());

	vector<dbInfo>::iterator it;
	vector<std::pair<Mat,Mat>>::iterator itYaml;

	map<int,string> MWWTestMapList;
	map<double,string> MWWTestWeightMapList;

	for ( it=vecListFile.begin(),itYaml = vqDataAllImage.begin()  ; it !=
vecListFile.end(); it++, itYaml++ )
	{
		string finalPath = it->yamlPath;

		Mat centerCluster;
		Mat eachClusterCount;
		yamlWR.readDataVqConst( finalPath.c_str()
,centerCluster,eachClusterCount);

		int MWWTestVaule = 0;
		double MWWTestWeightVaule = 0.0f;
		MWWTestCompute(sampleCenterCluster, centerCluster,
sampleEachClusterCount, eachClusterCount, MWWTestVaule, MWWTestWeightVaule);
		MWWTestMapList[MWWTestVaule] = it->imagePath;
		MWWTestWeightMapList[MWWTestWeightVaule] = it->imagePath;
	}

	int i =0;
	if (flagChooseMWW)
```

```
	{
		map<double,string>::reverse_iterator ritDouble;
		// show content:
		for ( ritDouble=MWWTestWeightMapList.rbegin() ; ritDouble !=
MWWTestWeightMapList.rend() && i<10 ; ritDouble++ ,i++)
			cout << ritDouble->first << ";" << ritDouble->second << endl;
		return 1;
	}
	map<int,string>::reverse_iterator ritInt;
	// show content:
	for ( ritInt=MWWTestMapList.rbegin() ; ritInt != MWWTestMapList.rend() &&
i<10; ritInt++ ,i++)
		cout << ritInt->first << ";" << ritInt->second << endl;
	return 1;
}

vector<dbInfo> readDataDbMysql(dbConnect dbConn,string DBID)
{
	vector<dbInfo> vecResult;
	Database db(dbConn.dbHost,dbConn.dbUser,dbConn.dbPass,dbConn.dbDBName);


	Query q(db);

	if (!db.Connected() || !q.Connected())
		return vecResult;
	// retrieve data
	string sql = "SELECT IDImage,imagePath,yamlPath from filedata";
	if (DBID.compare("")!=0)
	{
		sql = "SELECT IDImage,imagePath,yamlPath from filedata WHERE DBID =
"+ DBID;
	}
	q.get_result(sql);

	while (q.fetch_row())
	{
		dbInfo tmpAdd;
		tmpAdd.Id        = q.getval();
		tmpAdd.imagePath = q.getstr();
		tmpAdd.yamlPath  = q.getstr();
		vecResult.push_back(tmpAdd);
	}
	q.free_result();
	return vecResult;
}

void MWWTestCompute(Mat & sampleMat,Mat & compareMat,Mat &samplePopulaWeight,Mat
& comparePopulaWeight,int &MWWTestVaule,double & MWWTestWeightVaule)
{
	if (sampleMat.rows != compareMat.rows)
		return;

	int num_nodesMat         = sampleMat.rows + compareMat.rows;
	int num_EdgeCompleteMat  = (num_nodesMat*(num_nodesMat-1))/2;

	//first index of vertex
	//first index of vertex too
	typedef std::pair < int, int > EMat;

	//first index of all vertex
	//second type of each vertex
	//typedef std::pair < int, int > VMat;
	vector<EMat> allEdgesMat(num_EdgeCompleteMat);
```

```
	vector<int>   vertexInput(num_nodesMat);
	vector<Vec3f> vertexInputVec3f(num_nodesMat);
	vector<float> vertexInputWeightf(num_nodesMat);

	MatIterator_<Vec3f> sampleItf      = sampleMat.begin<Vec3f>();
	MatIterator_<Vec3f> sampleItf_end  = sampleMat.end<Vec3f>();
	MatIterator_<Vec3f> compareItf     = compareMat.begin<Vec3f>();
	MatIterator_<Vec3f> compareItf_end = compareMat.end<Vec3f>();

	MatIterator_<Vec<float, 1>> sampleWeightItf       =
samplePopulaWeight.begin<Vec<float, 1>>();
	MatIterator_<Vec<float, 1>> compareWeightItf      =
comparePopulaWeight.begin<Vec<float, 1>>();

	int countVertex = 0;

	for(; sampleItf != sampleItf_end; ++sampleItf,++sampleWeightItf) {
		vertexInput[countVertex]                = 0;
		vertexInputVec3f[countVertex]           = *sampleItf;
		vertexInputWeightf[countVertex]         = (*sampleWeightItf)[0];
		countVertex++;
	}
	for (; compareItf != compareItf_end; ++compareItf,++compareWeightItf){
		vertexInput[countVertex]                = 1;
		vertexInputVec3f[countVertex]           = *compareItf;
		vertexInputWeightf[countVertex]         = (*compareWeightItf)[0];
		countVertex++;
	}

	int countEdge   = 0;
	vector<double> weightsMat(num_EdgeCompleteMat);

	for(int i=0;i< num_nodesMat ;i++)
	{
		for(int j=i+1;j<num_nodesMat;j++)
		{
			allEdgesMat[countEdge].first  = i;
			allEdgesMat[countEdge].second = j;

			Vec3f tmpSrc  = vertexInputVec3f[i];
			Vec3f tmpDst  = vertexInputVec3f[j];

			double dim00 = (tmpSrc[0] - tmpDst[0]);
			double dim01 = (tmpSrc[1] - tmpDst[1]);
			double dim02 = (tmpSrc[2] - tmpDst[2]);

			weightsMat[countEdge] =
sqrt((dim00*dim00)+(dim01*dim01)+(dim02*dim02));
			countEdge++;
		}
	}
	typedef adjacency_list < vecS, vecS,
undirectedS,property<vertex_distance_t,double>, property < edge_weight_t,
double > > Graph;

	Graph g(num_nodesMat);
	property_map<Graph, edge_weight_t>::type weightmap = get(edge_weight, g);

	int sizeOfEdge = num_EdgeCompleteMat;

	for (std::size_t j = 0; j < sizeOfEdge ; ++j) {
		graph_traits<Graph>::edge_descriptor e;
		bool inserted;
```

```
            boost::tie(e, inserted) = add_edge(allEdgesMat[j].first,
allEdgesMat[j].second, g);
            weightmap[e] = weightsMat[j];
      }

      std::vector < graph_traits < Graph >::vertex_descriptor >
p(num_vertices(g));
      property_map<Graph, vertex_distance_t>::type distance =
get(vertex_distance, g);
      property_map<Graph, vertex_index_t>::type indexmap     = get(vertex_index,
g);
      prim_minimum_spanning_tree(g, *vertices(g).first, &p[0], distance,
weightmap, indexmap,default_dijkstra_visitor());

      int MWWValueInner = 0;
      for (std::size_t i = 0; i != p.size(); ++i)
            (p[i] != i) && (vertexInput[i] != vertexInput[p[i]]) ?
MWWValueInner++ : 0 ;

      vector<vector<int>> parentNode(num_nodesMat);
      for (std::size_t i = 0; i != p.size(); ++i)
          if (p[i] != i ){
                parentNode[p[i]].push_back(i);
          }
      float MWWTestWeightInner = 0.0f;
      for (int i=0 ; i<num_nodesMat ;++i){
            // (term00 + term01) / term02
            // term00 = a0 + a1
            // term01 = b0 + b1
            // term02 = a0+a1+b0+b1
            int tmpComp = vertexInput[i];
            if (tmpComp == 0){
                  float term00 = vertexInputWeightf[i];
                  float term01 = 0.0f;
                  float term02 = vertexInputWeightf[i];
                  vector<int> tempInner = parentNode[i];
                  if (tempInner.size() > 0)
                  {
                        for (int j =0; j < tempInner.size();j++){
                              float tmpValue = vertexInputWeightf[tempInner[j]];
                              term02 += tmpValue;
                              if (vertexInput[tempInner[j]] == tmpComp){
                                    term00 += tmpValue;
                              }else{
                                    term01 += tmpValue;
                              }
                        }
                        if(term02>0.0f)
                              MWWTestWeightInner += (term00*term01)/term02;
                  }

            }
      }
      MWWTestVaule       = MWWValueInner ;
      MWWTestWeightVaule = MWWTestWeightInner;
      return;
}
```

7.3 โปรแกรม Web Page สำหรับเก็บภาพลงในฐานข้อมูลภาพ

7.3.1 DBCommand.php

```
<?php
session_start();
$info = pathinfo($_SERVER['REQUEST_URI']);
$path="http://".$_SERVER['SERVER_NAME'].$info['dirname'];
if (empty($info['extension']))
        $path.="/".$info['basename'];
?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta  http-equiv="Content-Type" content="text/html; charset=utf-8" />
<link  href="fileuploader.css" rel="stylesheet" type="text/css" media="all" />
<title>Untitled Document</title>
<style type="text/css">
      #DBSelect { font-family: AngsanaUPC;
                        font-size: 30px;
                        font-weight: bolder;
                        color: #00F;
                        background-color: #CCC;
                        text-align: center;
                        margin: 0px;
                        padding: 0px;
                      }
      #CreateDatabase {
                        font-family: AngsanaUPC;
                        font-size: 30px;
```

```
                font-weight: bolder;
                color: #00F;
                background-color: #CCC;
                text-align: center;
                margin: 0px;
                padding: 0px;
                border-top-style: solid;
                border-right-style: solid;
                border-bottom-style: solid;
                border-left-style: solid;
            }
#SelectDB {
                text-align: right;
                font-family: AngsanaUPC;
                font-size: 30px;
                font-weight: bolder;
            }
#Image #InputImage {
                font-family: AngsanaUPC;
                font-size: 30px;
                font-weight: bolder;
                color: #00F;
                background-color: #CCC;
                text-align: center;
                margin: 0px;
                padding: 0px;
            }
#apDiv1 {
                position:absolute;
                left:290px;
```

```
                top:820px;
                width:494px;
                height:155px;
                z-index:1;
            }
        #JqPostForm fieldset legend {
                text-align: left;
            }
</style>
</head>

<body>
<label for="jumpmenu"></label>
<div>
<div id="DBSelect">
<form id="JqPostForm">
<fieldset>
<legend>เลือกฐานข้อมูล</legend>
<form id="form2" name="form2" method="post" action="" >
<select name="datafolder" id="datafolder" style="width:150px"
onchange="location.href='DBCommand.php?select='+this.value;">


<?php
@session_start();
session_unset();
mysql_connect("localhost","root","") or die(mysql_error());
mysql_select_db("TCBIR");
$query="SELECT DBID,Name FROM groupdatabase";
$result = mysql_query($query);
$checkValue = $_GET['select'];
while($nt=mysql_fetch_array($result))
```

```
{
	if ($checkValue == $nt[Name] || $checkValue =="" )
	{
		echo "<option selected value = $nt[Name]>$nt[Name]</option>";
		$checkValue = $nt[Name];
		$checkValueID = $nt[DBID];
	}else
	{
		echo "<option value = $nt[Name]>$nt[Name]</option>";
	}
}
?>
</select>
</form>
<fieldset>
<p>ป้อนภาพลงในฐานข้อมูล </p>

<div id="file-uploader">
<script src="fileuploader.js" language="javascript1.1"></script>
<script>
function createUploader()
{
	var uploader = new qq.FileUploader({
	//  pass the dom node (ex. $(selector)[0] for jQuery users)
	element: document.getElementById('file-uploader'),
	// path to server-side upload script. In our case server/php.php
	action: 'php.php',
	// additional data to send, name-value pairs
	params: {
		dataBaseSelect:  <?php echo "'"."$checkValue"."'"  ?>,
```

```
            dataBaseSelectID:  <?php echo "'"."$checkValueID"."'"  ?>
        }
        //debug: true
    });
}
window.onload = createUploader;
</script>
<script type="text/javascript">
var img_id=0
var image = new Array()
document.getElementById('send').onclick=function()
{
        img_id++
        var id="imgid"+img_id
        image = document.getElementById('file-uploader').value;
        document.getElementById('div').innerHTML="<img id='"+id+"' src='"+image+"'
width=500px height=200px>"
}
</script>
</div>
</fieldset>
</form>
<fieldset>
<div id="CreateDatabase">
<div>
<p>สร้างฐานข้อมูลใหม่ </p>
<form id="form1" name="form1" method="post" action="Newtable.php">
<p>
<label for="Namedatabase">สร้างฐานข้อมูล :    </label>
<input type="text" name="Namedatabase" id="Namedatabase" />
</p>
```

```
<p>
<label for="Detailofdatabase">ข้อมูลของฐานข้อมูล :</label>
<textarea name="Detailofdatabase" id="Detailofdatabase" cols="45"
rows="5"></textarea>
</p>
<p>
<input type="submit" name="submit" id="submit" value="ยืนยันการสร้าง" />
</p>
</form>
<p> </p>
</form>
</fieldset>
</body>
</html>
```

7.3.2 Php.php

```
<?php
require_once('Connections/conn.php');
session_start();
/**
* Handle file uploads via XMLHttpRequest
*/
class qqUploadedFileXhr {
/**
* Save the file to the specified path
* @return boolean TRUE on success
*/
function save($path) {
$input = fopen("php://input", "r");
$temp = tmpfile();
$realSize = stream_copy_to_stream($input, $temp);
```

```
fclose($input);
if ($realSize != $this->getSize()){
      return false;
}
$target = fopen($path, "w");
fseek($temp, 0, SEEK_SET);
stream_copy_to_stream($temp, $target);
fclose($target);
return true;
}

function getName() {
return $_GET['qqfile'];
}
function getSize() {
if (isset($_SERVER["CONTENT_LENGTH"])){
      return (int)$_SERVER["CONTENT_LENGTH"];
} else {
      throw new Exception('Getting content length is not supported.');
}
}
}

/**
* Handle file uploads via regular form post (uses the $_FILES array)
*/
class qqUploadedFileForm {
/**
* Save the file to the specified path
* @return boolean TRUE on success
```

```
*/
function save($path) {
if(!move_uploaded_file($_FILES['qqfile']['tmp_name'], $path)){
      return false;
}
return true;
}

function getName() {
return $_FILES['qqfile']['name'];
}

function getSize() {
return $_FILES['qqfile']['size'];
}
}

class qqFileUploader {
private $allowedExtensions = array();
private $sizeLimit = 10485760;
private $file;

function __construct(array $allowedExtensions = array(), $sizeLimit = 10485760){
$allowedExtensions = array_map("strtolower", $allowedExtensions);

$this->allowedExtensions = $allowedExtensions;
$this->sizeLimit = $sizeLimit;

$this->checkServerSettings();
```

```
if (isset($_GET['qqfile'])) {
//do this part
      $this->file = new qqUploadedFileXhr();
} elseif (isset($_FILES['qqfile'])) {
      $this->file = new qqUploadedFileForm();
} else {
      $this->file = false;
}
}

private function checkServerSettings(){
$postSize = $this->toBytes(ini_get('post_max_size'));
$uploadSize = $this->toBytes(ini_get('upload_max_filesize'));

if ($postSize < $this->sizeLimit || $uploadSize < $this->sizeLimit){
      $size = max(1, $this->sizeLimit / 1024 / 1024) . 'M';
      die("{'error':'increase post_max_size and upload_max_filesize to
$size'}");
}
}

private function toBytes($str){
$val = trim($str);
$last = strtolower($str[strlen($str)-1]);
switch($last) {
      case 'g': $val *= 1024;
      case 'm': $val *= 1024;
      case 'k': $val *= 1024;
}
return $val;
}
```

```
/**
* Returns array('success'=>true) or array('error'=>'error message')
*/
function handleUpload($uploadDirectory, $replaceOldFile = FALSE){
if (!is_writable($uploadDirectory)){
      return array('error' => "Server error. Upload directory isn't writable.");
}

if (!$this->file){
      return array('error' => 'No files were uploaded.');
}

$size = $this->file->getSize();

if ($size == 0) {
      return array('error' => 'File is empty');
}

if ($size > $this->sizeLimit) {
      return array('error' => 'File is too large');
}

$pathinfo = pathinfo($this->file->getName());

//filename
$filename = $pathinfo['filename'];
//echo $filename ;
//$filename = $pathinfo['filename'];
//$filename = md5(uniqid());
$ext = $pathinfo['extension'];
```

```
if($this->allowedExtensions && !in_array(strtolower($ext), $this->allowedExtensions)){
	$these = implode(', ', $this->allowedExtensions);
	return array('error' => 'File has an invalid extension, it should be one of '. $these . '.');
}

if(!$replaceOldFile){
	/// don't overwrite previous files that were uploaded
	while (file_exists($uploadDirectory . $filename . '.' . $ext)) {
		$filename .= rand(10, 99);
	}
}

try {
	/*** connect to SQLite database ***/
	$hostname_conn = "localhost";
	$database_conn = "ae";
	$username_conn = "root";
	$password_conn = "";
	$dbh = new PDO("mysql:host=$hostname_conn;dbname=$database_conn", $username_conn, $password_conn);
	$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
	$dbh->beginTransaction();

	$DBIDName = $_GET['dataBaseSelect'];
	$DBIDvalueID = $_GET['dataBaseSelectID'];
	$fullFileNameWithPath = $uploadDirectory . $filename . '.' . $ext;

	$yamlFilePath = "yaml/". $filename . '.yaml';
```

```
        $count = $dbh->exec("INSERT INTO filedata(DBID,imagePath,yamlPath)
VALUES('$DBIDvalueID','$fullFileNameWithPath','$yamlFilePath')");
if ($this->file->save($uploadDirectory . $filename . '.' . $ext)){
        $return_var = 0;
        $out = array();
        $exec = exec("VqKmeanCalculate.exe ".$fullFileNameWithPath."
".$yamlFilePath . " 60 30",$out,$return_var);

        if($return_var < 0)
        {
            unlink($fullFileNameWithPath);
            $dbh->rollback();
            $dbh = null;
            return array('error'=> 'Could not produce yaml file.');
        }
        $resultBool = $dbh->commit();
        $dbh = null;
        return array('success'=>true);
} else {
        $dbh->rollback();
        $dbh = null;
        return array('error'=> 'Could not save uploaded file.' .
            'The upload was cancelled, or server error encountered');
}
}
catch(PDOException $e)
{
        $dbh->rollback();
        $dbh = null;
}
}
```

```
}

$allowedExtensions = array();
// max file size in bytes
$sizeLimit = 10 * 1024 * 1024;

$uploader = new qqFileUploader($allowedExtensions, $sizeLimit);

$DBIDName = $_GET['dataBaseSelect'];
$Class2 =  "Image/allMix/";
$result = $uploader->handleUpload($Class2);
echo htmlspecialchars(json_encode($result), ENT_NOQUOTES);
?>
```

7.4 โปรแกรม Web Page สำหรับค้นหาภาพในฐานข้อมูล

7.4.1 SearchInner.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>class.upload.php test forms</title>
<style>
fieldset {
      width: 50%;
      margin: 15px 0px 25px 0px;
      padding: 15px;
}
legend {
      font-weight: bold;
}
.button {
```

```
	text-align: right;
}
.button input {
	font-weight: bold;
}

body {
	background-color: #F96;
}
</style>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<h1>Color Image Retrieval System</h1>
<fieldset>
<legend>Image sample</legend>
<p>Pick up image to upload and press upload </p>
<form name="form2" enctype="multipart/form-data" method="post"
action="upload.php" />
<p><input type="file" size="32" name="my_field" value="" /></p>
<p class="button"><input type="hidden" name="action" value="image" />
<input type="submit" name="Submit" value="upload to query" /></p>
</form>
<p> </p>
<p> </p>
</fieldset>
</body>
</html>
```

7.4.2 Retrieval.php

```
<?php

session_start();

error_reporting(E_ALL);


// recieve namedata form select to query each folder

$DBIDValue = "null" ;

// we first include the upload class, as we will need it here to deal with the
uploaded file

include('class.upload.php');

// retrieve eventual CLI parameters

$cli = (isset($argc) && $argc > 1);

if ($cli) {

      if (isset($argv[1])) $_GET['file'] = $argv[1];

      if (isset($argv[2])) $_GET['dir'] = $argv[2];

      if (isset($argv[3])) $_GET['pics'] = $argv[3];

}


// set variables

$dir_dest = (isset($_GET['dir']) ? $_GET['dir'] : 'sampleImage');

$dir_pics = (isset($_GET['pics']) ? $_GET['pics'] : $dir_dest);


if (!$cli) {

?>


<!DOCTYPE    html    PUBLIC    "-//W3C//DTD    XHTML    1.0    Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<meta http-equiv=Content-Type content="text/html; charset=utf-8">

<head>
```

```
<title>Color Image Retrieval System</title>
<style>
body {
}
fieldset {
	width: 50%;
	background: url(bg.gif);
	margin: 15px 0px 25px 0px;
	padding: 15px;
}
legend {
	font-weight: bold;
}
fieldset img {
	float: right;
}
fieldset p {
	font-size: 70%;
	font-style: italic;
}
.button {
	text-align: right;
}
.button input {
	font-weight: bold;
}
</style>
</head>

<body bgcolor="#FF6666">
```

```
<h1>Color Image Retrieval System :: viewing images retrieve</h1>
<?php
}
if ((isset($_POST['action']) ? $_POST['action'] : (isset($_GET['action']) ?
$_GET['action'] : '')) == 'image') {
// ---------- IMAGE UPLOAD ----------
// we create an instance of the class, giving as argument the PHP object
// corresponding to the file field from the form
// All the uploads are accessible from the PHP object $_FILES
$handle = new Upload($_FILES['my_field']);
// then we check if the file has been uploaded properly
// in its *temporary* location in the server (often, it is /tmp)
if ($handle->uploaded) {
// yes, the file is on the server
// below are some example settings which can be used if the uploaded file is an
image.
// now, we start the upload 'process'. That is, to copy the uploaded file
// from its temporary location to the wanted location
// It could be something like $handle->Process('/home/www/my_uploads/');
$handle->Process($dir_dest);
// we check if everything went OK
if ($handle->processed) {
      // everything was fine !
      echo '<fieldset>';
      echo '  <legend>sample file uploaded with success</legend>';
      echo '  <img src="'.$dir_pics.'/' . $handle->file_dst_name . '" />';
$info = getimagesize($handle->file_dst_pathname);
      echo '    link to the file : <a href="'.$dir_pics.'/' . $handle-
>file_dst_name . '">' . $handle->file_dst_name . '</a><br/>';
      echo '</fieldset>';
} else {
```

```
        // one error occured
        echo '<fieldset>';
        echo '  <legend>file not uploaded to the wanted location</legend>';
        echo '  Error: ' . $handle->error . '';
        echo '</fieldset>';
}

// we delete the temporary files
$handle-> Clean();

$return_var = 0;
$out = array();
$exec = exec("VqRetrieval.exe ".$dir_pics.'/' . $handle->file_dst_name . " 60 30
1 127.0.0.1 root null ae $DBIDValue",$out,$return_var);
if($return_var > 0)
        foreach ($out as $line) {  // process array line by line
                $pieces = explode(";", $line);
                $fileSplitPath = explode("/", $pieces[1]);
                $countData =  count($fileSplitPath);
                echo '<fieldset>';
                echo '  <legend>Retrieve image score := '.$pieces[0].'</legend>';
                echo '  <img src="'. $pieces[1] . '" />';
                $info = getimagesize($handle->file_dst_pathname);
                echo '    link to the file : <a href="'.$pieces[1] . '">' .
$fileSplitPath[$countData-1] . '</a><br/>';
                echo '</fieldset>';
        }
} else {
        echo '<fieldset>';
        echo '  <legend>file not uploaded on the server</legend>';
        echo '  Error: ' . $handle->error . '';
```

```
        echo '</fieldset>';
}
}

if (!$cli) {
        echo '<p><a href="Seachpage.html">do another test</a></p>';
?>
</body>
</html>
<?php
}
?>
```