```
//
//  main.cpp
//  Erasure Code
//
//  Created by Jerry Wang on 2017/11/21.
//  Copyright © 2017年 Jerry Wang. All rights reserved.
//
#include <iostream>
#include "functions.h"


int main(){

    dataInitial();
    dataPartition();
    dataDeploy();

    simulator();


    return 0;
}
```

```
//
//  functions.h
//  Erasure Code
//
//  Created by Jerry Wang on 2017/11/21.
//  Copyright © 2017年 Jerry Wang. All rights reserved.
//

#ifndef functions_h
#include "List.h"
#include "dataBlock.hpp"
#define functions_h
#define COL 6
#define ROW 4
#define Random(x) (float)(rand()%x)

#define DATA_NUM 24
#define BLOCK_NUM 4
#define ECBLOCK_NUM 2

extern float ** metadataMatrix;
extern float ** encodeMatrix;
extern float ** encodedDataMatrix;
extern float encodeMatrixData[BLOCK_NUM+ECBLOCK_NUM][BLOCK_NUM];

//数据准备工作，包括初始化数据和数据分片，元数据编码
//数据初始化，随机生成数据，写入metadata.txt
void dataInitial();
//数据分片，从文件读取
void dataPartition();
//数据分片预处理，格式化从文件读取的数据
void dataPrepare(List &L,char * buffer);
```

```
//数据编码
void erasureCode();
//数据部署
void dataDeploy();
//清除元数据矩阵，编码之后的数据矩阵，保留编码矩阵



//矩阵运算
void printMatrix(float **m1,int row,int col);
//void printMatrix(double ** m1,int row,int col);
float ** addMatrix(float **m1,float **m2,int row,int col);
float ** multiplyMatrix(float **m1,float **m2,int row,int col1,int col2);
float ** reverseMatirx(float **m1,int row,int col);
void deleteMatrix(float **m1,int row);
//int ** multiplyMatrix_double_int(float **m1,float **m2,int row);

//模拟服务器宕机（文件丢失）
void simulator();

//解码过程
//解码调用方法，入口参数：失效数据块号
void recovery(List &tagList);
//获取幸存的数据，构成数据矩阵，并获取解码矩阵M
void serializeData(List &tagList);
//依据数据矩阵和解码矩阵，求出原始数据矩阵，重构metadataMatrix
void decodeToMetaData();
//编码元数据矩阵，构成完整的数据矩阵
void  encodeMetaData();


void testSplit();

struct dataBlock{
    int blockNun;
    int * data;
};


#endif /* functions_h */



//
//  List.h
//  Erasure Code
//
//  Created by Jerry Wang on 2017/11/21.
//  Copyright © 2017年 Jerry Wang. All rights reserved.
//

#ifndef List_h
#define List_h

#define MAX_SIZE 100
#define INCREMENT_SIZE 10
```

```
struct List {
    float * elem;
    int listsize = 0;
    int length = 0;
};

//List dataList;

void initList(List &L);
void addElem(List &L, float elem);
void deleteElem(List &L, float position);
void readList(List &L);
float getElem(List &L, int position);

#endif /* List_h */


//
//  ListOperation.cpp
//  Erasure Code
//
//  Created by Jerry Wang on 2017/11/21.
//  Copyright © 2017年 Jerry Wang. All rights reserved.
//


#include <iostream>
#include <stdlib.h>
#include "List.h"

using namespace std;

void initList(List &L){
    L.elem = (float *)malloc(sizeof(float)*MAX_SIZE);
    L.listsize=MAX_SIZE;
    L.length=0;
}
void addElem(List &L,float elem){
    if (L.length>=L.listsize) {
        L.elem = (float *)realloc(L.elem, sizeof(float)*(INCREMENT_SIZE+L.listsize));
    }
    L.elem[L.length]=elem;
    L.length++;
}
void deleteElem(List &L, int position){
    for (int i = position; i<L.length-1; i++) {
        L.elem[i]=L.elem[i+1];
    }
    L.length--;
}
void readList(List &L){
    for (int i=0; i<L.length; i++) {
        cout<<getElem(L, i)<<endl;
    }
}
float getElem(List &L, int position){
    return L.elem[position];
}
```

```
//
//  dataBlock.hpp
//  Erasure Code
//
//  Created by Jerry Wang on 2017/11/22.
//  Copyright © 2017年 Jerry Wang. All rights reserved.
//

#ifndef dataBlock_hpp
#define dataBlock_hpp

#include <iostream>
#include "List.h"
using namespace std;

struct DataBlock{
    int tag;
    List dataSet;
};

void initDataBlock(DataBlock & db,int tag,int dataNum,List &metadata);

#endif /* dataBlock_hpp */


//
//  dataBlock.cpp
//  Erasure Code
//
//  Created by Jerry Wang on 2017/11/22.
//  Copyright © 2017年 Jerry Wang. All rights reserved.
//

#include "dataBlock.hpp"

void initDataBlock(DataBlock & db,int tag,int dataNum,List &metadata){

    db.tag = tag;
    initList(db.dataSet);
    for (int i =6*(tag); i<6*(tag+1); i++) {
        addElem(db.dataSet, metadata.elem[i]);
    }

}




//
//  matrixOperation.cpp
//  Erasure Code
//
//  Created by Jerry Wang on 2017/11/22.
//  Copyright © 2017年 Jerry Wang. All rights reserved.
//

#include <iostream>
using namespace std;
```

```
void printMatrix(float **m1,int row,int col){
    for (int i=0; i<row; i++) {
        for (int j=0; j<col; j++) {
            cout<<m1[i][j]<<" ";
        }
        cout<<endl;
    }
}
float ** addMatrix(float **m1,float **m2,int row,int col){
    float ** result ;
    result = new float* [row];
    for (int i=0; i<row; i++) {
        result[i] = new float[col];
    }
    for(int i=0;i<row;i++){
        for (int j=0; j<col; j++) {
            result[i][j]=m1[i][j]+m2[i][j];
        }
    }
    return result;
}
float ** multiplyMatrix(float **m1,float **m2,int row,int col1,int col2){
    float ** result ;
    result = new float* [row];
    for (int i=0; i<row; i++) {
        result[i] = new float[col2];
    }
    for (int i=0; i<row; i++) {
        for (int j=0; j<col2; j++) {
            result[i][j]=0;
            for (int k=0; k<col1; k++) {
                result[i][j]+=m1[i][k]*m2[k][j];
            }
        }
    }

    return result;
}
void deleteMatrix(float **m1,int row){
    for (int i=row-1; i>=0; i--) {
        delete [] m1[i];
    }
    delete [] m1;
}

float ** reverseMatirx(float **m1,int row,int col){
    float ** result;

    float ** L = new float *[row];
    float ** L_reverse = new float *[row];
    float ** U = new float *[row];
    float ** U_reverse = new float *[row];
    //1.初始化L，U矩阵
    for(int i = 0;i<row;i++){
        L[i]=new float[row];
        L_reverse[i] = new float[row];
        L[i][i]=1.0;//下三角单位矩阵
        U[i]=new float[row];
        U_reverse[i] = new float[row];
```

```
}
//2.求U矩阵第一行
for (int i =0; i<row; i++) {
    U[0][i]=m1[0][i];
}
//3.求L矩阵第一列
for (int i = 1; i<row; i++) {
    L[i][0]=m1[i][0]/U[0][0];
}
for (int i =1 ; i<row;i++) {
    //4.求U矩阵的i行元素
    for (int j=i; j<row; j++) {
        float sum =0.0;
        for (int k =0; k<i; k++) {
            sum += (L[i][k]*U[k][j]);
        }
        U[i][j] = m1[i][j]-sum;
    }
    //5.求L矩阵的i列元
    for (int j = i+1; j<row; j++) {
        float sum = 0.0;
        for (int k = 0; k<i; k++) {
            sum += (L[j][k]*U[k][i]);
        }
        L[j][i] = (m1[j][i]-sum)/U[i][i];
    }
    for (int j = 1; j<row; j++) {
        if (j>i) {
            L[i][j]=0.0;
            U[j][i]=0.0;
        }
    }
}
cout<<"L 矩阵："<<endl;
printMatrix(L, row, row);
cout<<"U 矩阵："<<endl;
printMatrix(U, row, row);

cout<<"LU 矩阵的积："<<endl;
printMatrix(multiplyMatrix(L, U, row,row,row), row, row);

//求L，U矩阵的逆矩阵
for (int i =0; i<row; i++) {
    L_reverse[i][i]=1.0/L[i][i];
    U_reverse[i][i]=1.0/U[i][i];
}
//6.求L矩阵的逆矩阵
for (int i = 1; i<row; i++) {
    for (int j = 0; j<row-i; j++) {
        float sum =0.0;
        for (int k = i; k>0; k--) {
            sum += (L[j+k][i-k]*L_reverse[i-k][j]);
        }
        L_reverse[i+j][j]=-sum;
    }
}
cout<<"L 矩阵的逆矩阵："<<endl;
```

```
    printMatrix(L_reverse, row, row);
    //7.求U矩阵的逆矩阵
    for (int i = 1; i<row; i++) {
        for (int j = 0; j<row-i; j++) {
            float sum = 0.0;
            for (int k = 1; k<= i; k++) {
                sum += (L[j][k+j]*U_reverse[k+j][i+j]);
            }
            U_reverse[j][j+i] = -sum/L[j][j];
        }
    }
    cout<<"U 矩阵的逆矩阵： "<<endl;
    printMatrix(U_reverse, row, row);

    result = multiplyMatrix(U_reverse, L_reverse, row,row,row);


    return result;

}



//
//  prepare.cpp
//  Erasure Code
//
//  Created by Jerry Wang on 2017/11/21.
//  Copyright © 2017年 Jerry Wang. All rights reserved.
//

#include <iostream>
#include <fstream>
#include <stdlib.h>
#include "functions.h"
using namespace std;

#define MEATDATA "metadata.txt"

List dataList;
DataBlock dataBlocks[4];
DataBlock ecBlocks[2];
float ** metadataMatrix;
float ** encodeMatrix;
float ** encodedDataMatrix;
float encodeMatrixData[BLOCK_NUM+ECBLOCK_NUM][BLOCK_NUM] = {
    {1,0,0,0},
    {0,1,0,0},
    {0,0,1,0},
    {0,0,0,1},
    {1,1,1,1},
    {1,2,4,8}
};

//随机生成原始数据，分割为4*6的原始数据块
void dataInitial(){
    ifstream file(MEATDATA);
    if(!file){
        ofstream file(MEATDATA);
```

```
        for (int i=0; i<24; i++) {
            file<<Random(10)<<" ";
        }
        file.close();
    }
    else{
        cout<<"元数据已创建"<<endl;
    }
    metadataMatrix = new float * [BLOCK_NUM];
    for (int i = 0; i<BLOCK_NUM; i++) {
        metadataMatrix[i]=new float[DATA_NUM/BLOCK_NUM];
    }
    encodeMatrix = new  float * [BLOCK_NUM+ECBLOCK_NUM];
    for (int i = 0; i<BLOCK_NUM+ECBLOCK_NUM; i++) {
        encodeMatrix[i]= new float [BLOCK_NUM];
        for (int j=0; j<BLOCK_NUM; j++) {
            encodeMatrix[i][j]=encodeMatrixData[i][j];
        }
    }
}

void dataPrepare(List &L, char * buffer){
    const char * split = " ";
    char * value = strtok(buffer, split);
    while (value!=NULL) {
        float metadata;
        sscanf(value, "%f",&metadata);
        addElem(L, metadata);
        value = strtok(NULL, split);
    }
}

void dataPartition(){
    //将元数据从文件中读出，并转换为int数组,存储于dataList中
    ifstream file(MEATDATA);
    char * buffer;
    if (!file.is_open()) {
        cout<<"文件打开失败！"<<endl;
        exit(0);
    }
    else{
        initList(dataList);
        while (!file.eof()) {
            buffer = new char[256];
            file.getline(buffer, 256);
            dataPrepare(dataList, buffer);
        }
    }

    //数据分片，以4个数据块为例,并构建元数据矩阵（4*6）
    const int dataNum =dataList.length/BLOCK_NUM;
    for (int i = 0; i<BLOCK_NUM; i++) {
        initDataBlock(dataBlocks[i], i, dataNum, dataList);
        for (int j=0; j<dataNum; j++) {
            metadataMatrix[i][j]=dataBlocks[i].dataSet.elem[j];
        }
    }
    cout<<"元数据矩阵："<<endl;
```

```
    printMatrix(metadataMatrix, BLOCK_NUM, DATA_NUM/BLOCK_NUM);

    encodedDataMatrix = new float *[BLOCK_NUM+ECBLOCK_NUM];
    for (int i =0; i<BLOCK_NUM+ECBLOCK_NUM; i++) {
        encodedDataMatrix[i]=new float [DATA_NUM/BLOCK_NUM];
    }

    encodedDataMatrix = multiplyMatrix(encodeMatrix, metadataMatrix,
(BLOCK_NUM+ECBLOCK_NUM), BLOCK_NUM, (DATA_NUM/BLOCK_NUM));

    cout<<"编码数据：  "<<endl;
    printMatrix(encodedDataMatrix, BLOCK_NUM+ECBLOCK_NUM, DATA_NUM/BLOCK_NUM);

    for (int i =0;i<ECBLOCK_NUM; i++) {
        ecBlocks[i].tag= BLOCK_NUM+i+1;
        initList(ecBlocks[i].dataSet);
        for (int j=0; j<DATA_NUM/BLOCK_NUM; j++) {
            addElem(ecBlocks[i].dataSet, encodedDataMatrix[i][j]);
        }
    }
}

//保存数据段在不同的文件中，使用tag命名

void dataDeploy(){
    for (int i =0; i<BLOCK_NUM+ECBLOCK_NUM; i++) {
        char* filename;
        filename = new char [1];
        filename[0] = char((int)'0'+i);
        ifstream infile(filename);
        if (infile) {
            cout<<"文件已存在，准备删除并重新创建数据文件"<<endl;
            remove(filename);
        }
        ofstream outfile(filename);
        for (int j=0; j<DATA_NUM/BLOCK_NUM; j++) {
            outfile<<encodedDataMatrix[i][j]<<" ";
        }
        outfile.close();
    }
    deleteMatrix(metadataMatrix, BLOCK_NUM);
    deleteMatrix(encodedDataMatrix, BLOCK_NUM+ECBLOCK_NUM);
}



//
//  decoded.cpp
//  Erasure Code
//
//  Created by Jerry Wang on 2017/11/23.
//  Copyright © 2017年 Jerry Wang. All rights reserved.
//

#include <iostream>
#include <fstream>
#include "functions.h"
using namespace std;
```

```
float ** correctDataMatrix;
float ** decodeCheckMatrix;

float ** decodeMatrix_2_3;
float ** decodeMatrix_1_5;
float matrixData_2_3[BLOCK_NUM][BLOCK_NUM] = {
    {1,0,0,0},
    {-1.5,2,2,-0.5},
    {0.5,-3,-1,0.5},
    {0,1,0,0}
};
float matrixData_1_5[BLOCK_NUM][BLOCK_NUM] = {
    {-2,-4,-8,1},
    {1,0,0,0},
    {0,1,0,0},
    {0,0,1,0}
};
int judge = 0;


void initDecodeMatrix(){
    decodeMatrix_1_5 = new float *[BLOCK_NUM];
    decodeMatrix_2_3 = new float *[BLOCK_NUM];
    for (int i = 0 ; i<BLOCK_NUM; i++) {
        decodeMatrix_1_5[i] = new float[BLOCK_NUM];
        decodeMatrix_2_3[i] = new float[BLOCK_NUM];
        for (int j = 0; j<BLOCK_NUM; j++) {
            decodeMatrix_1_5[i][j] = matrixData_1_5[i][j];
            decodeMatrix_2_3[i][j] = matrixData_2_3[i][j];
        }
    }
}


void serializeData(List &tagList){

    if(getElem(tagList, 0)>=BLOCK_NUM||getElem(tagList, 1)>=BLOCK_NUM)
        judge=1;
    else
        judge=0;
    correctDataMatrix = new float*[BLOCK_NUM];
    decodeCheckMatrix = new float * [BLOCK_NUM];
    for(int i=0;i<BLOCK_NUM;i++){
        correctDataMatrix[i] = new float[DATA_NUM/BLOCK_NUM];
        decodeCheckMatrix[i] = new float[BLOCK_NUM];
    }
    int block=0;
    //读取数据，形成数据矩阵和编码矩阵
    for (int i =0; i<BLOCK_NUM+ECBLOCK_NUM; i++) {
        if (i != getElem(tagList, 0) && i != getElem(tagList, 1)) {
            List temp;
            initList(temp);
            char * filename = new char [4];
            filename[0] = (char) ((int)'0'+i);
            ifstream infile(filename);
            if(!infile.is_open()){
                cout<<"文件打开失败"<<endl;
                exit(0);
            }
            while (!infile.eof()) {
```

```
            char * buffer = new char[256];
            infile.getline(buffer, 256);
            dataPrepare(temp, buffer);
        };
        for (int j = 0 ;j<DATA_NUM/BLOCK_NUM; j++) {
            correctDataMatrix[block][j] = getElem(temp, j);
        }
        for (int j = 0; j<BLOCK_NUM; j++) {
            decodeCheckMatrix[block][j] = encodeMatrixData[i][j];
        }
        remove(filename);
        block++;
    }
  }

  cout<<"编码矩阵："<<endl;
  printMatrix(decodeCheckMatrix, BLOCK_NUM, BLOCK_NUM);
  cout<<"正常数据矩阵："<<endl;
  printMatrix(correctDataMatrix, BLOCK_NUM, DATA_NUM/BLOCK_NUM);

}
void decodeToMetaData(){
  initDecodeMatrix();
  metadataMatrix = new float * [BLOCK_NUM];
  for (int i = 0 ; i<BLOCK_NUM; i++) {
    metadataMatrix[i] = new float[DATA_NUM/BLOCK_NUM];
  }
  if (judge==1) {
    metadataMatrix = multiplyMatrix(decodeMatrix_1_5, correctDataMatrix,
BLOCK_NUM,BLOCK_NUM,DATA_NUM/BLOCK_NUM);
  }
  else{
    metadataMatrix = multiplyMatrix(decodeMatrix_2_3, correctDataMatrix,
BLOCK_NUM,BLOCK_NUM,DATA_NUM/BLOCK_NUM);
  }
  cout<<"恢复元数据："<<endl;
  printMatrix(metadataMatrix, BLOCK_NUM, DATA_NUM/BLOCK_NUM);
}
void encodeMetaData(){
  encodedDataMatrix = new float *[BLOCK_NUM+ECBLOCK_NUM];
  for (int i =0; i<BLOCK_NUM+ECBLOCK_NUM; i++) {
    encodedDataMatrix[i]=new float [DATA_NUM/BLOCK_NUM];
  }

  encodedDataMatrix = multiplyMatrix(encodeMatrix, metadataMatrix,
(BLOCK_NUM+ECBLOCK_NUM), BLOCK_NUM, (DATA_NUM/BLOCK_NUM));

  cout<<"编码数据："<<endl;
  printMatrix(encodedDataMatrix, BLOCK_NUM+ECBLOCK_NUM, DATA_NUM/BLOCK_NUM);

}
void recovery(List &tagList){

  serializeData(tagList);
  decodeToMetaData();
  encodeMetaData();
  dataDeploy();
}
```

```
//
//  simulator.cpp
//  Erasure Code
//
//  Created by Jerry Wang on 2017/11/23.
//  Copyright © 2017年 Jerry Wang. All rights reserved.
//

#include <iostream>
#include "functions.h"

void simulator(){
    List tagList;
    char * filename;
    //1.数据块和校验块分别丢失一个数据块
    initList(tagList);
    int dataBlock=0,ecBlock=0;
    addElem(tagList, dataBlock);
    addElem(tagList, ecBlock+BLOCK_NUM);
    //删除数据块对应的文件，模拟失效
    filename = new char[1];
    filename[0]=(char) ((int)'0'+dataBlock);
    remove(filename);
    delete [] filename;
    filename = new char[1];
    filename[0]=(char)((int)'0'+ecBlock+BLOCK_NUM);
    remove(filename);
    delete [] filename;

    recovery(tagList);

    //2.数据块丢失两个数据块
    initList(tagList);
    int * dataBlocks = new int[2];
    dataBlocks[0]=1;
    dataBlocks[1]=2;
    addElem(tagList, dataBlocks[0]);
    addElem(tagList, dataBlocks[1]);
    //删除数据块对应的文件，模拟失效
    for (int i =0; i<2; i++) {
        filename = new char[1];
        filename[0]=(char) ((int)'0'+dataBlocks[i]);
        remove(filename);
        delete [] filename;
    }

    recovery(tagList);
}
```