《实验3: 哈夫曼数编码》

实验报告

|  |  |  |
| --- | --- | --- |
| 工作分工 | 姓名 | 学号 |
| 编码实现 | 胡少卿 | 2020211825 |
| 测试 | 喻成卓 | 2020212457 |
| 报告编写 | 赵航 | 2020212797 |

2020年 12 月 7日

# 题目：实验题目

## 问题描述：

编写一个程序，实现如下功能：

1） 编码功能：输入一个文件 A，依据 A 文件的内容，对文件中的每个字节采用赫夫曼算法进行重新编码，并将编码所用的赫夫曼树和对文件内容重新编码的结果输出到另外一个文件 B 中。

2） 解码功能：输入编码后的文件 B，解码还原，将还原内容保存在文件 C

中，要求文件 A 和 C 内容完全一致。

## 设计描述

1. 输入提示：

首先在运行该程序之前，确认该次执行此程序是需要解码还是编码，提示用户输入，利用输入值判断是否用户所选择的模式。

1. 编码流程：
2. 首先统计该文件中存储字符出现次数，建立appear\_times数组并且使用边读入边计算的方式, 避免大文件保存不下。之后利用根据appear\_times数组统计得到的出现次数数组建立哈夫曼树并且根据根据哈夫曼树对字符进行编码，得到编码表，之后利用与**其他小组商定的辅助信息的格式**，构建本次输出的辅助信息。随后读入用户选择的模式，决定编码完成后输出比特流或字符串格式。并且，在输出编码之后的文件时同样采用边读入边输出的方式，从而有效避免大文件存不下而导致的问题
3. 解码流程：
4. 首先读入解码所需要的辅助信息，并且利用读入的辅助信息还原**编码者**所使用的哈夫曼树。并且确认读入的为字符还是比特流，在确认读入种类之后选择相应的方式进行解码
5. 具体算法流程简述：
6. 建立哈夫曼树，函数参有两者构成，weight和leaves，weight为各个字符出现的次数, leaves为字符数。首先初始化森林, 即为每个不同的字符建一棵树，之后将该将森林合并成为哈夫曼树，具体流程如下：取森林中最小的数和第二小的树，将这两棵树合并后的新树加入森林中，完成leaves-1次合并后唯一留下的树即为哈夫曼树。
7. 统计待编码文件中所有字符的出现次数。首先找到文件的结尾, 确定读入范围，之后边读入边计算，避免保存读入的文件而导致内存问题。
8. 二进制比特流输出。需要注意的是在输出已编码文件的过程中，如果用户所选择的是比特流输出，则需要额外输出有多少位有效的二进制比特位作为辅助信息，由print\_bits函数完成。
9. 辅助信息输出，根据与**其他小组商定的辅助信息的格式**，需要输出文件格式、被编码文件格式等信息。
10. 该程序中所定义的部分函数以及变量简述：

**unsigned long long appear\_times[MaxCharSize]; //存储字符出现次数**

**static Node newNode(Node \*node1, Node \*node2); //合并节点**

**bool is\_leave();//判断是否是叶子节点**

**char data() //返回节点代表的字符**

**Node \*left() { return lchild; } //返回节点的左儿子**

**Node \*right() { return rchild; } //返回字节的右儿子**

**int w() { return weight; } //返回节点的权重**

**static HuffmanTree BuildHT(unsigned long long \*weght);//建立哈夫曼树**

**static HuffmanTree newTree(int weight, char element);//搭建一颗只有一**

**个节点的新树**

**static HuffmanTree combine(HuffmanTree left, HuffmanTree right);//合并两棵树**

**Node \*root() { return head; }//返回根节点**

**int read\_runmode(); //读入运行模式,即编码或解码**

**int read\_printmode(); //读入编码模式, 即字符串或比特流**

**bool read\_filename(char \*infile, char \*outfile); //读入输入输出文件名**

**bool read\_string\_group(unsigned long long \*msg, char \*infile, int &mode);**

**//读入编码后文件的辅助信息**

**void print\_bits(unsigned long long \*appear\_times, char**

**(\*map)[MaxTreeDepth], char \*outfile);**

**//输出接下来编码文件中有多少个有效的二进制比特位**

**void print\_msg\_group(unsigned long long \*appear\_times, char \*outfile,**

**int mode); //输出编码的辅助信息**

**void print\_codes\_group(char \*infile, char (\*map)[MaxTreeDepth], char**

**\*outfile, int mode); //一边读入待编码内容一边输出编码的结果**

**void print\_codes\_group(HuffmanTree &HT, char \*infile, char \*outfile, int mode);**

**void encoding(Node\* root, char (\*coding\_table)[MaxTreeDepth], char code[], int t);//求编码表**

**bool count\_occchar\_group(unsigned long long \*appear\_times, char \*infile);//求待编码文件中各字符出现的次数**

1. 哈夫曼树使用的数据结构:

哈夫曼树由链表构成, Node为节点, HuffmanTree保存哈夫曼树的根节点, 具体如下所示:

class Node

{

private:

    unsigned int weight; //保存权重, 即字符出现的次数

    char element;        //保存代表的字符

    Node \*parent, \*lchild, \*rchild;

public:

Node(int weight = 0, char element = '\0', Node \*parent = NULL, Node \*lchild = NULL, Node \*rchild = NULL)

: weight(weight), element(element), parent(parent), lchild(lchild), rchild(rchild) {}

    static Node newNode(Node \*node1, Node \*node2); //合并节点

    bool is\_leave();                               //判断是否是叶子节点

    char data() { return element; }                //返回节点代表的字符

    Node \*left() { return lchild; }                //返回节点的左儿子

    Node \*right() { return rchild; }               //返回字节的右儿子

    int w() { return weight; }                     //返回节点的权重

};

class HuffmanTree

{

private:

    Node \*head;//哈夫曼树的根节点

public:

    HuffmanTree(Node \*head = NULL) : head(head) {}

    static HuffmanTree BuildHT(unsigned long long \*weght);

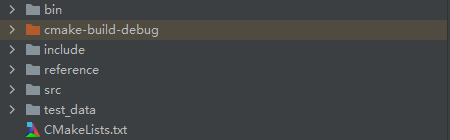
    static HuffmanTree newTree(int weight, char element);

    static HuffmanTree combine(HuffmanTree left, HuffmanTree right);

    Node \*root() { return head; }

};

1. 特殊说明
2. 合作小组：马志小组
3. 辅助信息说明：开头四个字节MAGIC\_NUMBER为MZPS, 代表文件格式;接下来一字节为bool变量, 0表示被编码为文本, 1表示被编码为二进制文件。接下来是256个八字节的unsigned long long变量, 保存每个字符的出现次数。如果为二进制文件, 接下来还有八位字节的unsigned long long变量保存接下来有多少个有效的二进制比特位(在print\_codes\_group中读入)
4. 项目文件结构说明：



**Bin：**编译完成后的可执行文件

**Include：**所使用的头文件

**Reference**：该次实验所使用的材料以及马志小组的可执行程序

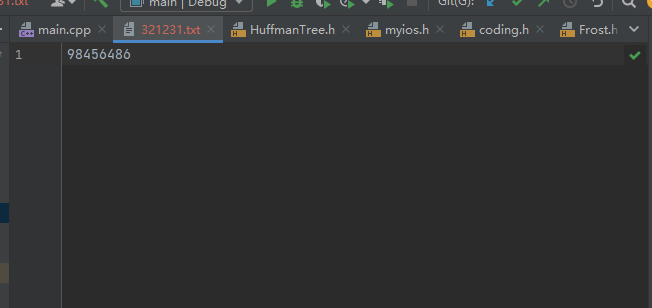
**Src：**所有的源代码文件，分为五个模块

**Test\_data:**部分用于测试的文件

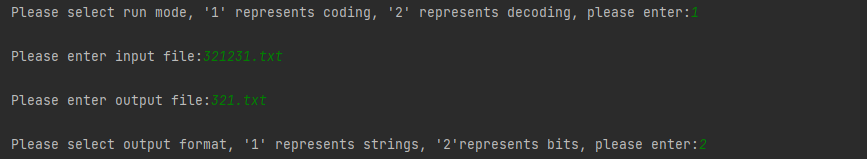
**Cmake-build-debug**:所使用的cmake产生的文件

## 测试设计

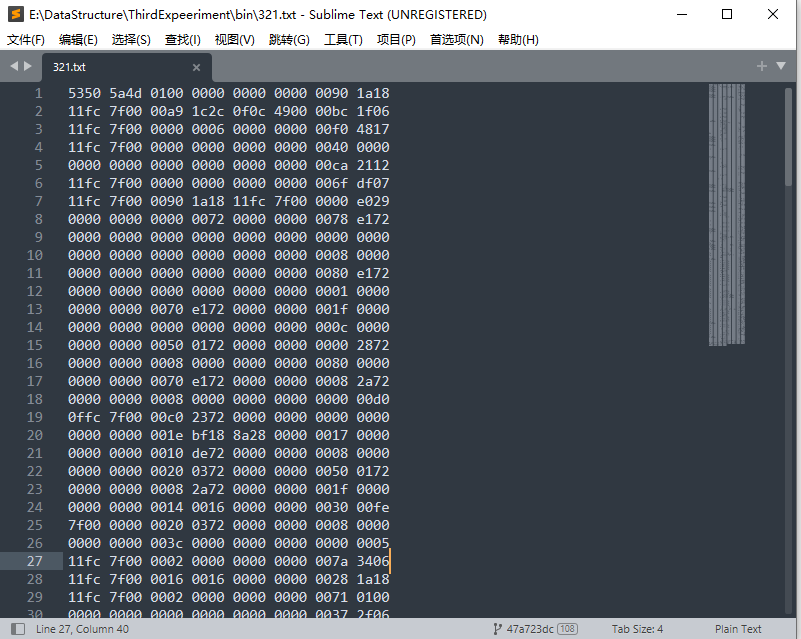
1. 文本文件的压缩与解压缩



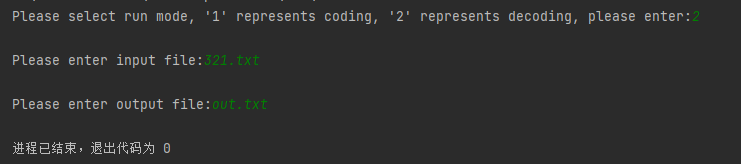
如图显示的内容，选择编码模式，比特流输出到321.txt



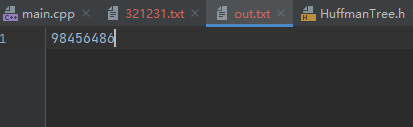
利用sublime打开321.txt，结果如下



发现达到设计所构想的要求。接下来测试解码，重新运行该程序，输入参数如下



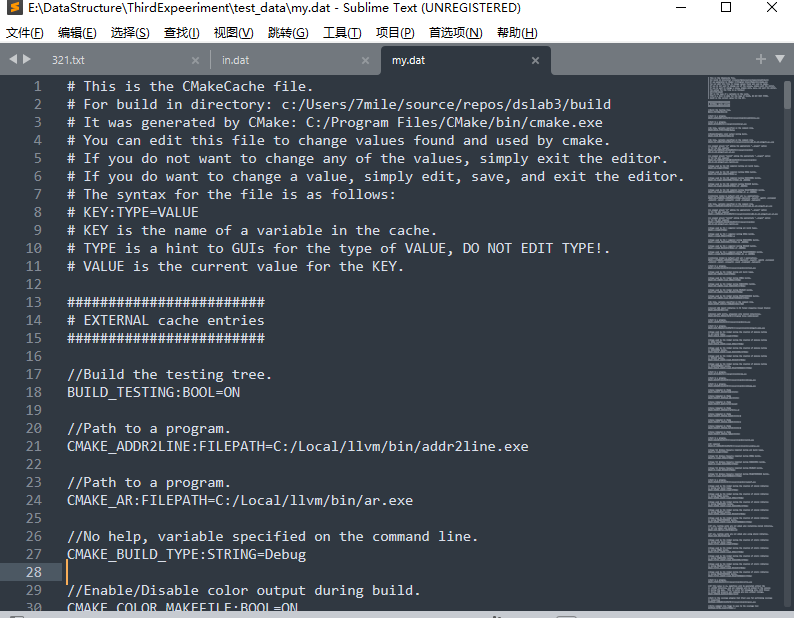
解码结果如下



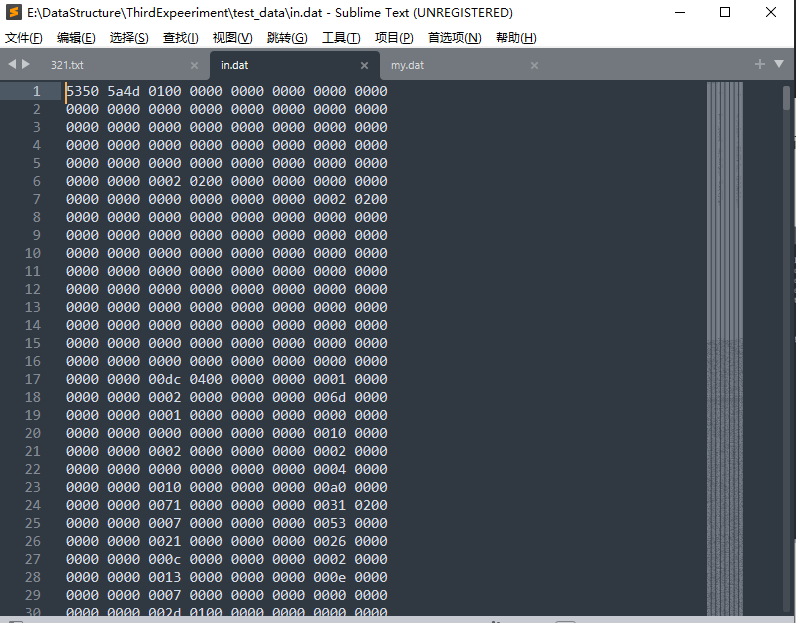
成功。

与上述相同操作，源文件，编码结果，解码结果如下（由于操作相同故不再展示命令输入等操作）

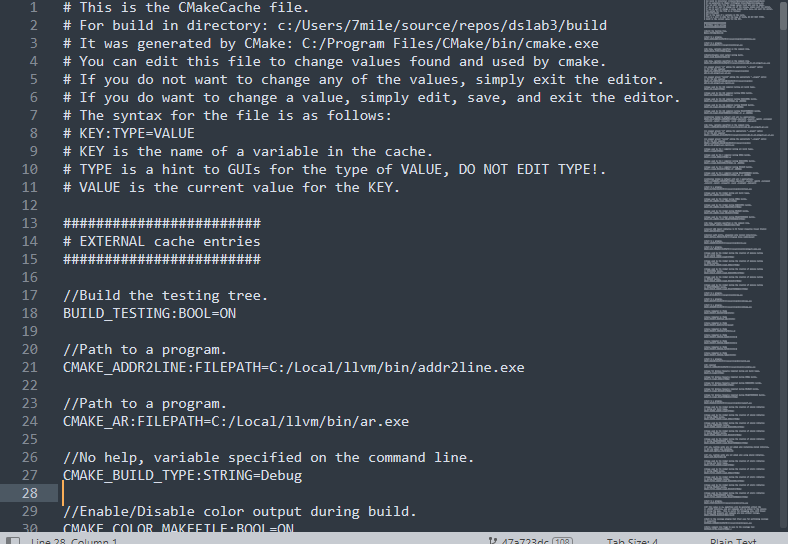
源文件



编码结果



解码结果



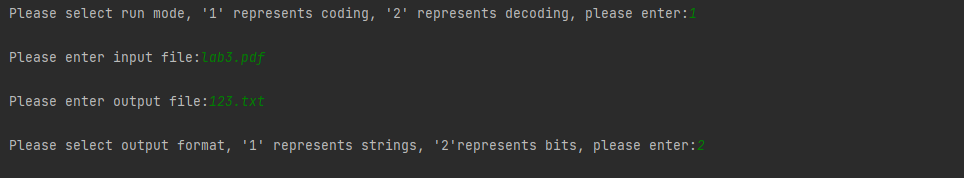
成功

1. 非文本文件

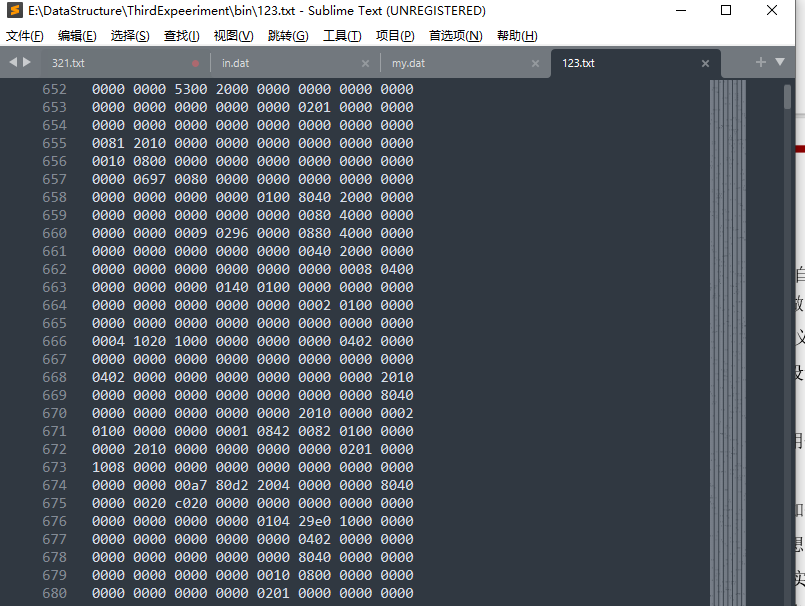
该部分的测试，我们小组选取了常见的各种文件格式如：pdf，docx，jpg，ppt等文件，并且充分考虑到了文件的大小，但由于篇幅有限，报告内仅展示所选测试内容的一部分。这里展示pdf文件的编码与解码，该次测试所选取的文件为lab3.pdf



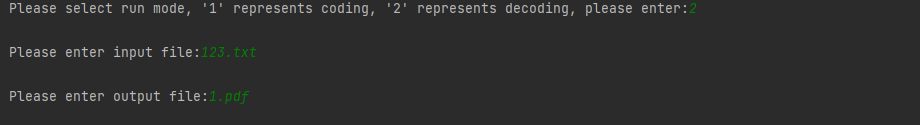
参数如下



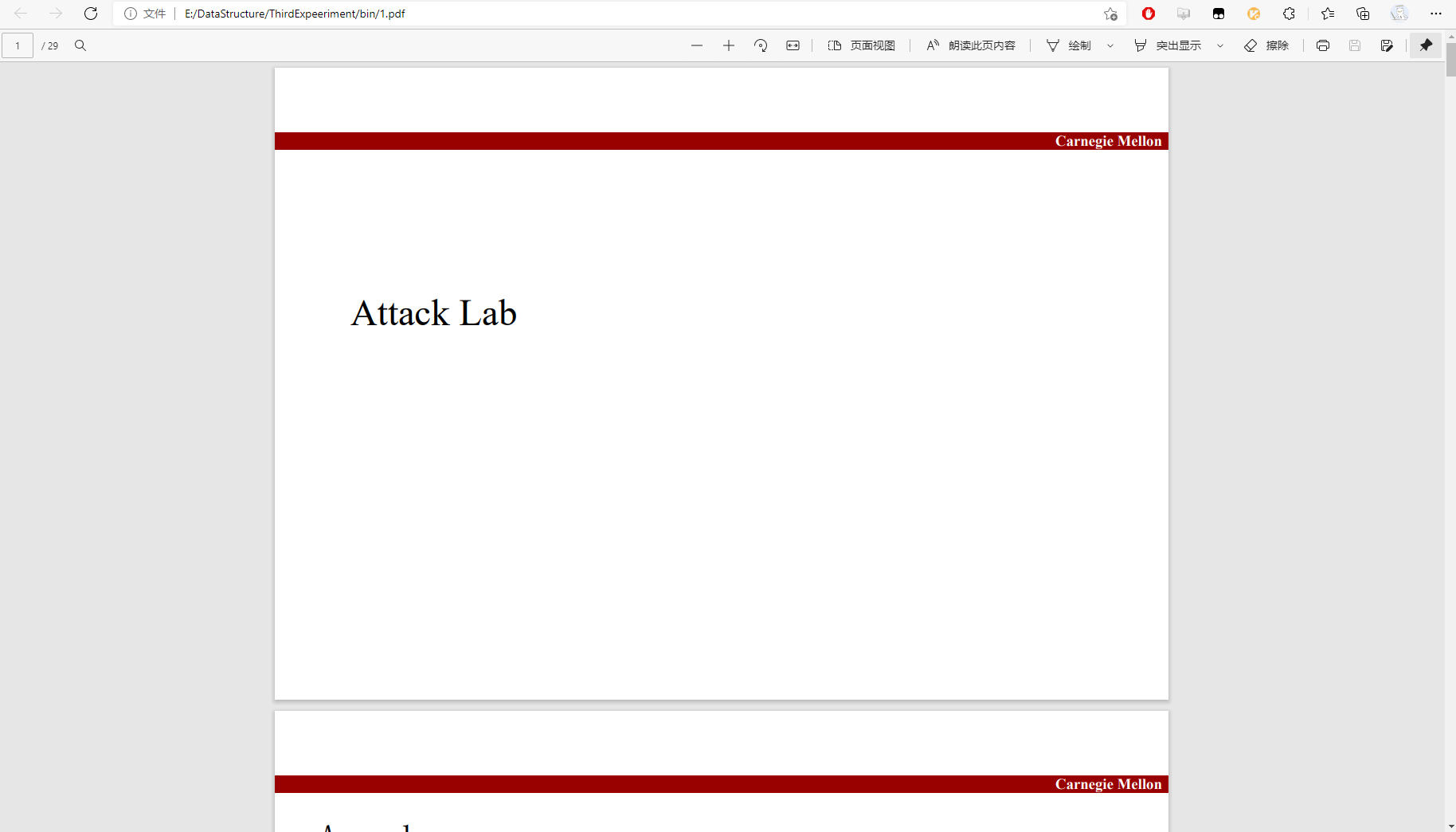
编码结果如下



解码参数如下



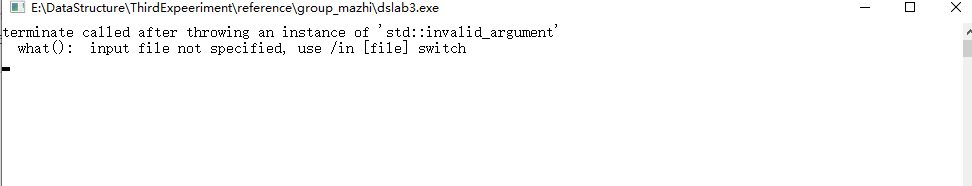
解码结果如下



成功

1. 跨小组编解码

本次实验中我们小组选取的合作小组为马志小组，在与马志小组商量之后，我们两组确定了辅助信息的格式，在双方完成后，互相交换可执行文件进行测试，马志小组的可执行文件在所提交的目录中ThirdExpeeriment\reference\group\_mazhi下，并且附带有使用的命令行命令。



利用马志小组的程序，将之前测试过的所有样例中选取所具有代表性的五例进行测试，发现能够解码我们小组所编码的文件，并且使用马志小组所编码的文件作为我们小组程序的输入文件也能够得到正确的结构，即可认为该次跨小组任务成功。

## 心得体会

在本次为期半个月的时间内，通过我们小组各成员之间的相互讨论和合作，我们完成了这次实验，更值得高兴的是我们完成了该次实验中的跨小组编码与解码任务，这对于我们的考验是前所未有的，虽然在之前的几次实验中，我们小组内部已经完成了多次实验，并且都取得了比较不错的结果，但是这次跨小组的合作却对我们是一次比较大的考验。借助这次的实验，我们小组第一次通过跨小组构思，和多位同学讨论并且不断尝试来设计共同的标准。这次设计，不仅巩固了我以前所学的知识，还让我对跨小组的合作更深一步的了解，掌握了更多的技巧和技能，以及对之后的跨小组合作有了一个简单的认识。我们清楚的认识到，在以后的学习和工作中不断要做好自己的事情，还要学会和其他人一起沟通交流。要高效完成任务，仅仅有高超的技能是不够的，还要学会合作，有较强的实践能力。只有多动手，经常编写程序，多合作，才能发现我们学习上的漏洞和自己的在合作交流不足，并在实践中解决这些问题，不断提高自己转化知识的能力。

## 提交材料说明

在之前的文件结构说明中以及说明上传文件的目录层次和作用