* **题目：将任意文件进行哈夫曼编码并进行压缩**
* **班级：2021211312 姓名：杨天宇 学号：2021211281** 分工： **5,6**
* **班级：2021211312 姓名：王昊钰 学号：2021211282** 分工： **1,2**
* **班级：2021211312 姓名：李志茂 学号：2021211289** 分工： **3,4**

1. 需求分析

输入输出形式：

文档中的字符均为扩充ASCII码表示。

在同级文件夹中放一个in.txt输出一个压缩后文件lock.txt

在同级文件夹中放一个lock.txt,输出一个解压文件lock.txt

程序能达到的功能

将一个文件读入，经过哈夫曼编码之后，对源文件进行压缩，输出压缩后的文件.

将一个压缩后的文件读入，解压缩后输出原文件。

输出结果

若正确压缩和解压，输出Coding success!

若无法解压该文件，输出Error:It is not file which I can understand.

1. 概要设计

解决思路：

压缩：先统计文档中不同字符出现次数作为权重，之后根据权重大小对这些字符进行哈夫曼编码，之后在文件开头输入元数据作为对压缩文件的标识，然后以一个字节为单位进行压缩并输出至目标文件中。

解压：读取文档开头的元数据检查是否可以解压，之后读取哈夫曼编码，对文件内容进行解压并输出，根据元数据中保存的最后有效输入长度处理最后一个字节。

数据结构类型：

哈夫曼树，哈夫曼树结点，用于生成哈夫曼编码

元数据结构，用于标识文件是否为程序所压缩文件，错误时的输出以及最后一个字节的数据。

字符频度表，用于统计各字符出现次数

主程序流程

压缩：初始化哈夫曼树与字符频度表，之后对文档进行哈夫曼编码。编码完成后首先向输出文件中输出元数据用于标识，之后输出压缩后的哈夫曼编码表，最后将所有内容压缩后输出到目标文件

解压：首先读取元数据确定是否能够解压，错误则输出错误提示。之后读取哈夫曼编码表，最后对压缩内容进行解压。

函数调用：

压缩：

调用initHtree和createHtree初始化哈夫曼树。

调用getHCode对文档进行哈夫曼编码

调用getlastValidBIt对最后一个字节进行特殊处理

解压：

调用initHtree和createHtree初始化哈夫曼树。

调用Hdecoding解压文件

1. 详细设计

数据类型：Node 树的结点结构 Htree\_head 元数据

Encipher.c:

typedef struct Node{

int weight,parent,lc,rc;

char \*code;

}Node;//树各节点的结构

typedef struct Htree\_head{

char flag[5];

char last;

char unused[10];

}Fhead; //元数据结构

void initHtree(Node \*ht)//树的初始化

{

for(int i=0;i<512;i++)

{

ht[i].weight=0;

ht[i].lc=-1;

ht[i].rc=-1;

ht[i].parent=-1;

ht[i].code = (char \*) calloc(sizeof(char),256);

}

}

int createHtree(Node \*ht,int \*Num)//创建哈夫曼树

{

unsigned char word;

FILE \*fp=fopen("in.txt","rb");

word=fgetc(fp);

while(!feof(fp))

{

ht[word].weight++;

Num[word]++;

word=fgetc(fp);

}

fclose(fp);

/\*串联树的各结点，按照哈夫曼编码\*/

int i,j,k,m;

for(i=256;i<511;i++)

{

j=-1,k=-1;

for(int l=0;l<i;l++)//寻找权最小的两个结点，j结点的权不大于k结点权

{

if(ht[l].parent==-1&&ht[l].weight!=0){

if(j==-1){

j=l;

}

else if(k==-1){

k=l;

if(ht[j].weight>ht[k].weight){

m=j;

j=k;

k=m;

}

}

else if(ht[l].weight<ht[j].weight){

k=j;

j=l;

}

else if(ht[l].weight<ht[k].weight)

k=l;

}

}

if(k==-1)//只剩下一个没有双亲的节点，又叫根

break;

ht[j].parent=i;

ht[k].parent=i;

ht[i].weight=ht[j].weight+ht[k].weight;

ht[i].lc=j;

ht[i].rc=k;

}

return j;

}

void getHCode(Node \*ht,int root,int index,char \*code)//生成哈夫曼编码

{

char Code[256];

strcpy(Code,code);

if(ht[root].lc!=-1&&ht[root].rc!=-1)

{

code[index]='0';

getHCode(ht,ht[root].lc,index+1,code);

strcpy(code,Code);

code[index]='1';

getHCode(ht,ht[root].rc,index+1,code);

strcpy(code,Code);

}

else{

strcpy(ht[root].code,code);

}

}

//取最后一个字节的有效位数

int getlastValidBit(Node \*ht) {

int sum = 0;

int i;

for(i = 0; i < 256; i++) {

sum += strlen(ht[i].code) \* ht[i].weight;

sum &= 0xFF;

}

sum &= 0x7;

return sum == 0 ? 8 : sum;

}

int main()

{

Node ht[511];

int root=-1;

int Num[256]={};

initHtree(ht);//树的初始化

root=createHtree(ht,Num);//创建树并返回根

char Code[256]={};//开辟空间用于保存哈夫曼编码

getHCode(ht,root,0,Code);//生成哈夫曼编码

FILE \*fpIn=fopen("in.txt","rb");

FILE \*fpOut=fopen("lock.txt","wb");

Fhead filehead;

strcpy(filehead.flag,"bupt");//标识符，判断是否为该程序生成的加密文件

filehead.last=getlastValidBit(ht);//文件最后一个字节的有效长度

fwrite(&filehead,sizeof(Fhead),1,fpOut);//输入元数据

fwrite(Num,sizeof(int),256,fpOut);//输入原文件各字符频度

/\*按位进行哈夫曼编码\*/

int index=0;

unsigned char word,value=0;

char \*code;

word=fgetc(fpIn);

while(!feof(fpIn)){

code=ht[word].code;

for(int i=0;code[i];i++){

if(code[i]=='0'){

value &= (~(1 << ((index) ^ 7))); //value的index位赋0

}else{

value |= (1 << ((index) ^ 7)); //value的index位赋1

}

index++;

if(index>=8)

{

index=0;

fwrite(&value,sizeof(char),1,fpOut);

}

}

word=fgetc(fpIn);

}

if(index)

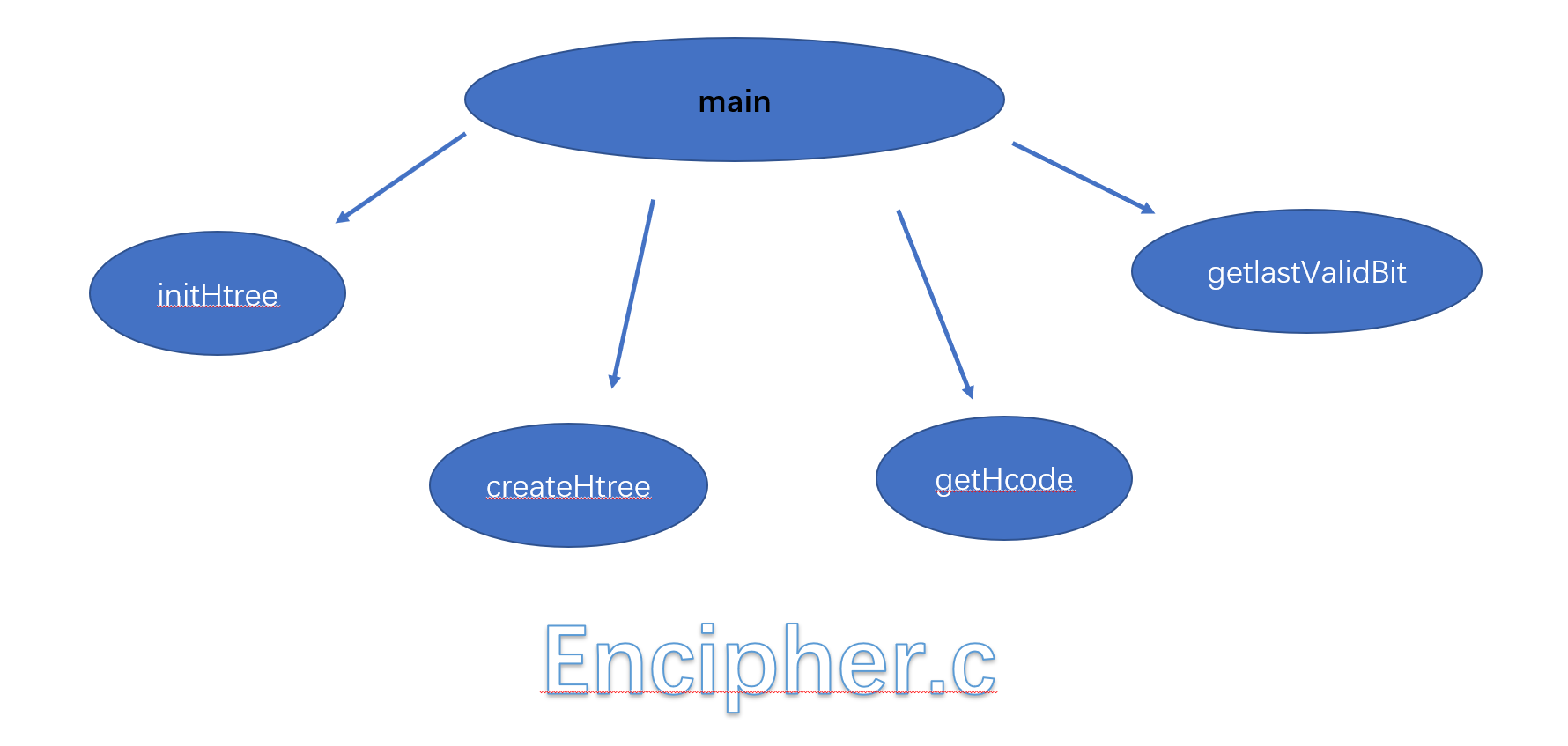
fwrite(&value,sizeof(char),1,fpOut);

fclose(fpIn);

fclose(fpOut);

return 0;

}



unlock.c:

结构体声明同上；

void initHtree(Node \*ht)//树的初始化

{

for(int i=0;i<512;i++)

{

ht[i].weight=0;

ht[i].lc=-1;

ht[i].rc=-1;

ht[i].parent=-1;

ht[i].code = (char \*) calloc(sizeof(char),256);

}

}

int createHtree(Node \*ht,int \*Num)//生成哈夫曼树

{

for(int i=0;i<256;i++){

ht[i].weight=Num[i];

} //由Num数组传回的字符频度赋给树的各元素weight

int i,j,k,m;

for(i=256;i<511;i++)//串起树的各元素

{

j=-1,k=-1;

for(int l=0;l<i;l++)

{

if(ht[l].parent==-1&&ht[l].weight!=0){

if(j==-1){

j=l;

}

else if(k==-1){

k=l;

if(ht[j].weight>ht[k].weight){

m=j;

j=k;

k=m;

}

}

else if(ht[l].weight<ht[j].weight){

k=j;

j=l;

}

else if(ht[l].weight<ht[k].weight)

k=l;

}

}

if(k==-1)//只剩下一个元素没有双亲，它不是根谁是

break;

/\*给i和j找双亲\*/

ht[j].parent=i;

ht[k].parent=i;

ht[i].weight=ht[j].weight+ht[k].weight;

ht[i].lc=j;

ht[i].rc=k;

}

if(k!=-1)

return i;

else

return j;//返回根节点

}

void getHCode(Node \*ht,int root,int index,char \*code)//生成哈夫曼码

{

char Code[256];//用于调用函数后恢复code

strcpy(Code,code);

if(ht[root].lc!=-1&&ht[root].rc!=-1)

{

code[index]='0';

getHCode(ht,ht[root].lc,index+1,code);

strcpy(code,Code);//code返回调用前

code[index]='1';

getHCode(ht,ht[root].rc,index+1,code);

strcpy(code,Code);//同上

}

else{

strcpy(ht[root].code,code);//将哈夫曼码传给树

}

}

void Hdecoding(Node \*ht,int root,char last)//解码函数

{

FILE \*fpIn=fopen("lock.txt","rb");

FILE \*fpOut=fopen("out.txt","wb");

fseek(fpIn,0L,SEEK\_END);//跳到文件尾

long fileSize = ftell(fpIn);//文件总长度

fseek(fpIn,16+sizeof(int)\*256,SEEK\_SET);//跳过元数据

long curLocation = ftell(fpIn);//记下当前位置

unsigned char value,valueOut;

int index=0,i=root,j;

fread(&value,sizeof(unsigned char),1,fpIn);

bool finished=false;

while(!finished)

{

if(ht[i].lc==-1&&ht[i].rc==-1){

valueOut=i;

fwrite(&valueOut,sizeof(unsigned char),1,fpOut);

i=root;

if(curLocation>=fileSize&&index>=last)

break;

}

j=(value & (1 << ((index) ^ 7)));

if(j==0)

i=ht[i].lc;

else

i=ht[i].rc;

if(++index>=8){

index=0;

if(curLocation>=fileSize){

break;//强制退出：避免因bug而出现死循环 （你可别往后读了啊！）

}

fread(&value,sizeof(unsigned char),1,fpIn);

curLocation=ftell(fpIn);

}

}

printf("Coding success！\n");//解码成功，奖励自己

fclose(fpIn);

fclose(fpOut);

}

int main()

{

FILE \*fpIn=fopen("lock.txt","rb");//打开加密文件

/\*识别元数据\*/

Fhead head;

int Num[256];

fread(&head,sizeof(Fhead),1,fpIn);//读取元数据

if(head.flag[0]!='b'||head.flag[1]!='u'||head.flag[2]!='p'||head.flag[3]!='t')//根据元数据判断是否为本程序可解码的加密文件

{

printf("Error:It is not file which I can understand.\n");//这个加密文件不是我所能解的，量力而行

return 0;

}

fread(Num,sizeof(int),256,fpIn);//读取加密文件中各字符频度，用于生成哈夫曼树和哈夫曼编码

fclose(fpIn);//有打开当然有关闭

/\*哈夫曼树、哈夫曼码的初始化\*/

Node ht[511];

int root,last=head.last;//此处为bug出现后的修补，数初始化后head数据会出错（数据溢出？）

initHtree(ht);//树的初始化

root=createHtree(ht,Num);//生成哈夫曼树并返回根节点

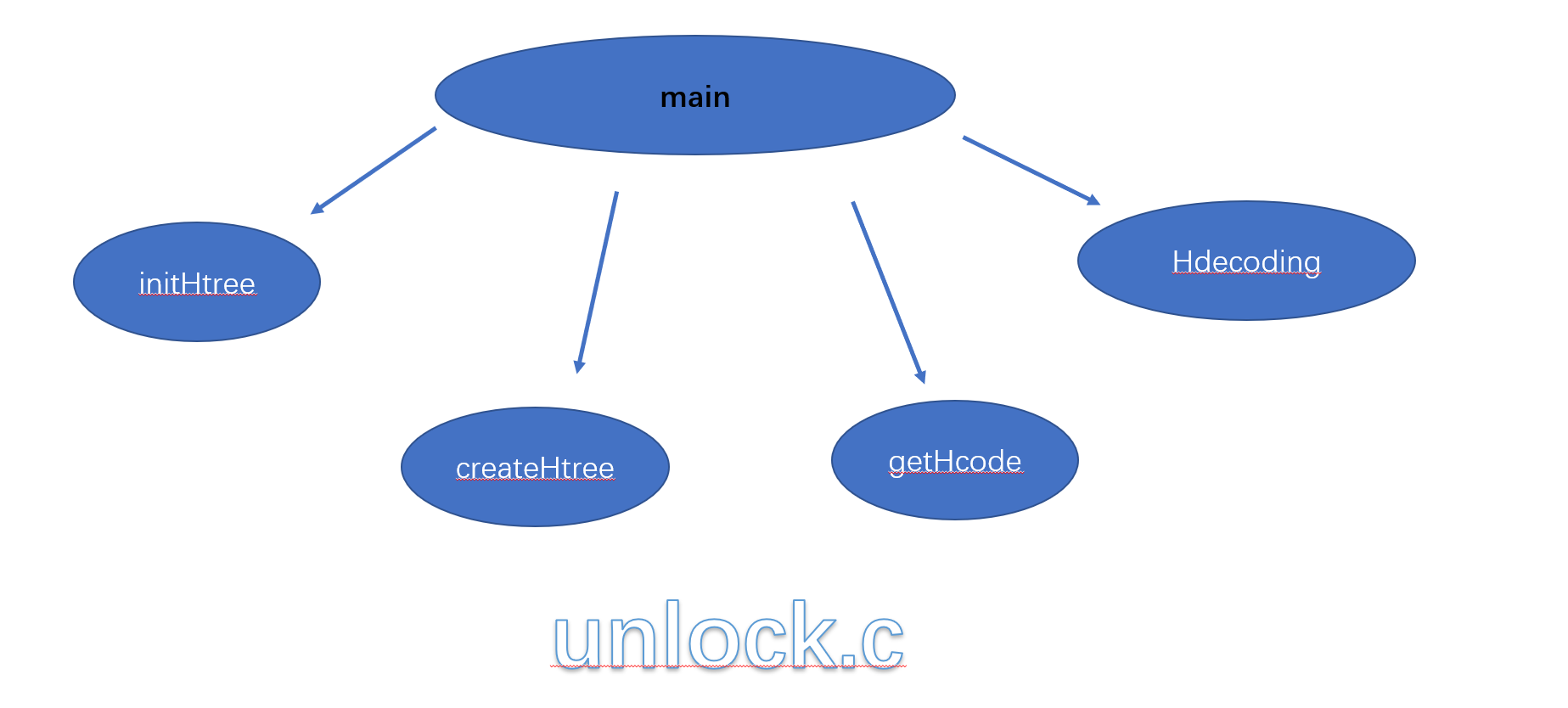
char code[256];//为每个字符开辟哈夫曼码的存储空间

getHCode(ht,root,0,code);//生成哈夫曼码并传入Node

Hdecoding(ht,root,last);//解码加密文件

return 0;

}



4、调试分析报告

遇到的问题：哈夫曼编码的生成、哈夫曼码生成后所编译产生的数据如何传入加密文件、如何解压加密文件；

解决方案:回顾梳理老师上课所介绍的内容以及PPT，再查阅相关书籍资料；

回顾讨论：通过编译利用哈夫曼码压缩文件及解压的程序加深对哈夫曼树及哈夫曼码的印象，对其应用有了更多的了解。

Encipher.c：时间复杂度：O(n2);空间复杂度：O（n）。

Unlock.c：时间复杂度：O(n2);空间复杂度：O（n）。

改进设想： 哈夫曼码的长度可以根据具体哈夫曼树的高度改变以节省空间、编译内容可以尝试拓展至中文字符

经验和体会：加深了对树和哈夫曼码的了解和掌握，对其运用也有了更多的了解

1. 用户使用说明

将对应压缩原文件放在程序同文件夹目录下，改名为in.txt，然后运行encipher.c，如果运行成功则程序会在同目录下生成lock.txt压缩文件;再运行unlock.c程序，即在同目录下生成lock.txt的解压缩文件out.txt.

1. 测试结果

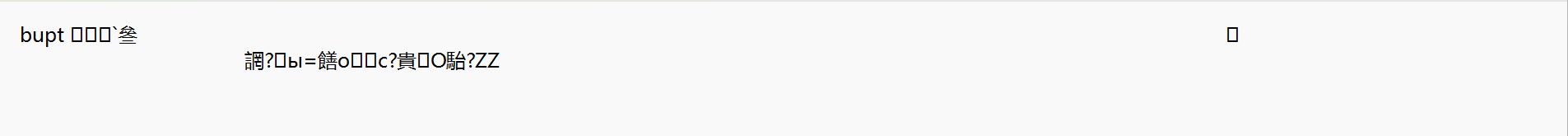
输入1：

in.txt:

Happiness can be found even in the darkest of times.

输出1:

lock.txt:



out.txt:

Happiness can be found even in the darkest of times.

输入2：

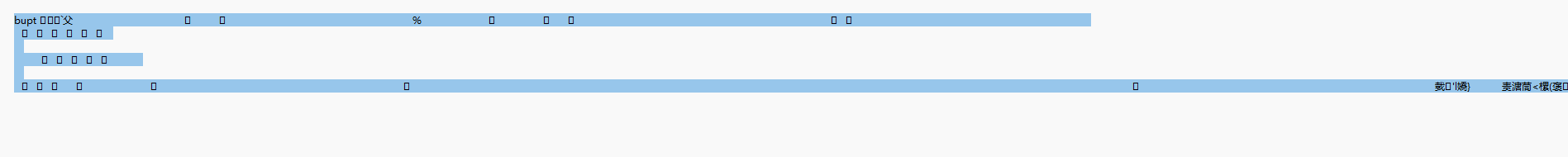
in.txt:

Happiness is not about being immortal nor having food or rights in one's hand.

It’s about having each tiny wish come true, or having something to eat when you are hungry or having someone's love when you need love.

输出2:

lock.txt:



out.txt:

Happiness is not about being immortal nor having food or rights in one's hand.

It’s about having each tiny wish come true, or having something to eat when you are hungry or having someone's love when you need love.