**基于哈夫曼编码的文件压缩与解压**

**实验题目：基于哈夫曼编码的文件压缩与解压**

|  |  |  |  |
| --- | --- | --- | --- |
| 班级：2020211305 | 姓名：缪奇志 | 学号：2020211345 | 分工：需求分析和概要设计 |
| 班级：2020211305 | 姓名：倪玮昊 | 学号：2020211346 | 分工：详细设计和调试分析报告 |
| 班级：2020211305 | 姓名：杨智杰 | 学号：2020211356 | 分工：用户使用说明和测试结果 |

1. **需求分析**

题目任务：

将任意一个指定文件进行哈夫曼编码，并以真正的二进制位生成一个二进制文件，反过来，可将一个压缩文件解码还原为原来的文件。

程序的输入和输出：

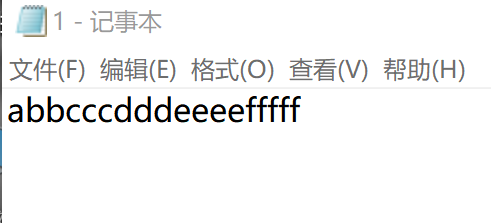
|  |  |  |  |
| --- | --- | --- | --- |
| 输入 | 数据类型 | 值的范围 | 代表意义 |
| sourcefile | 字符串 | 无 | 输入文件名 |
| encodefile | 字符串 | 无 | 命名的压缩文件的名字 |
| decodefile | 字符串 | 无 | 命名的解压文件的名字 |
| source | 字符串 | 无 | 待操作的源文件的内容 |
| dest | 字符串 | 无 | 待操作的目标文件的内容 |

程序功能实现（功能有4）：

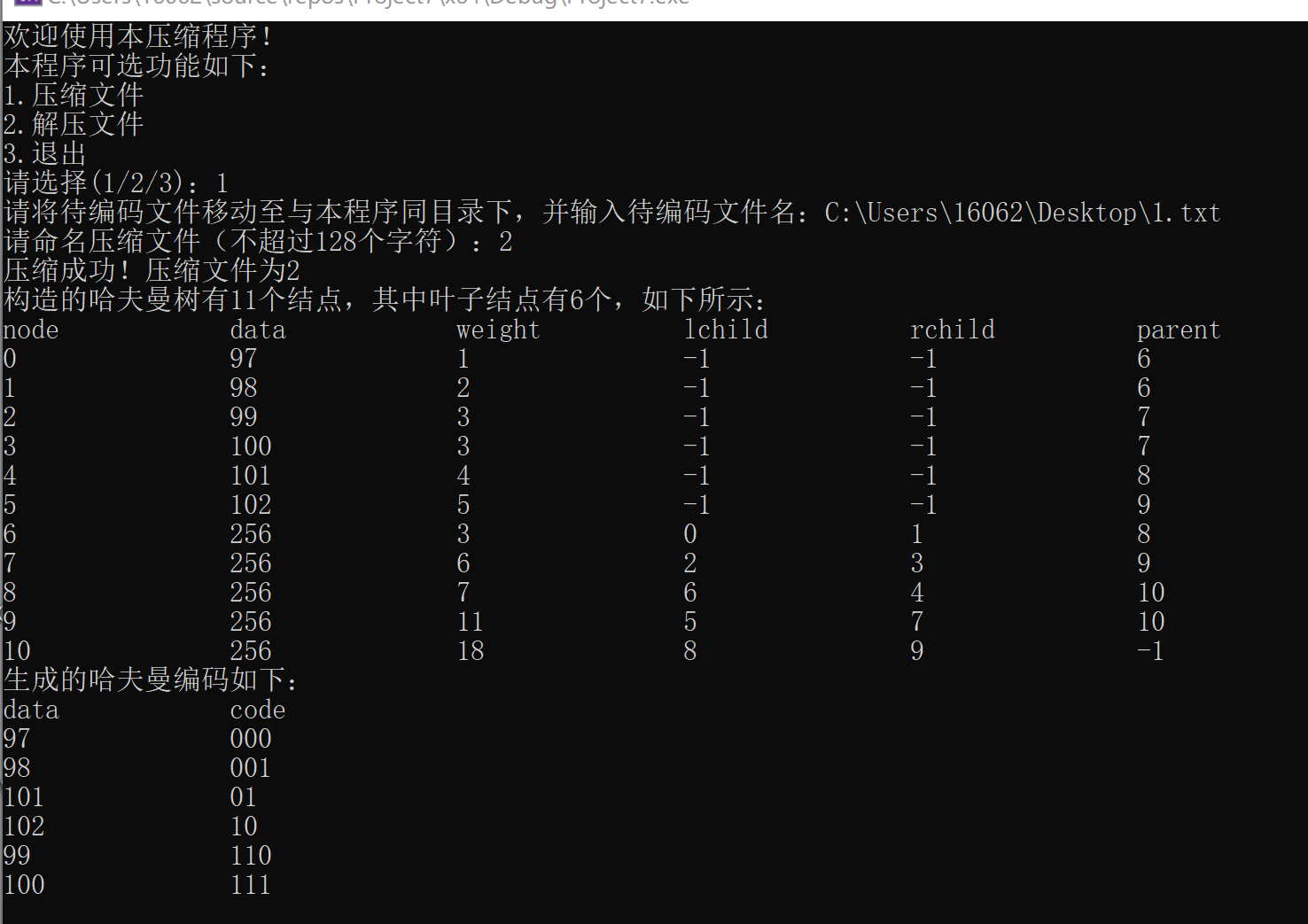
1. 功能 1：压缩文件
2. 功能2：解压文件
3. 功能3：退出程序
4. 其他情况：输出报错提醒

正确数据输出结果范例：

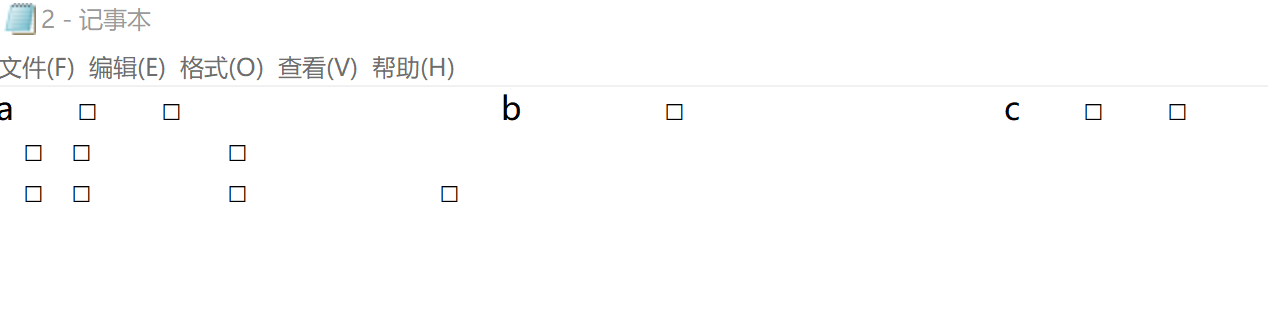
输入文件为：



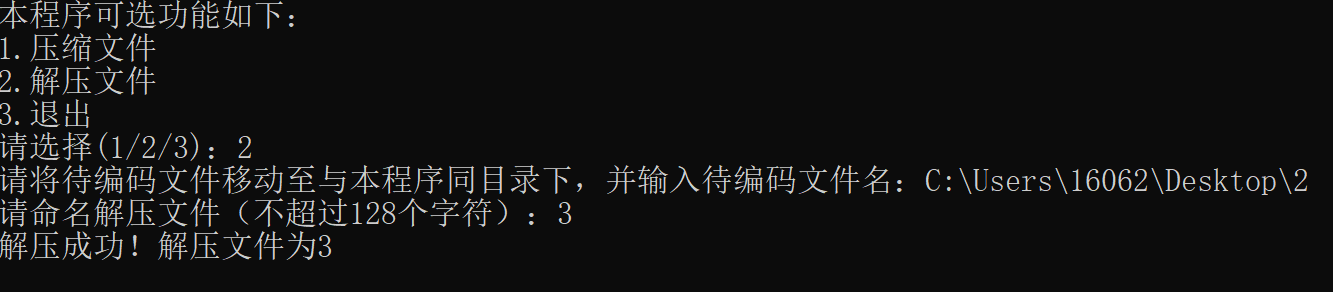
程序运行结果为：



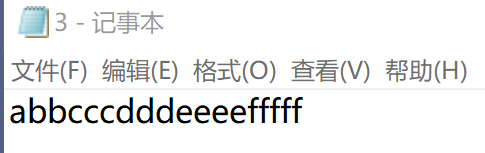
得到二进制文件打开为：



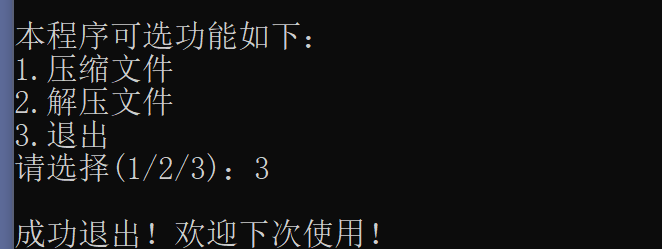
对二进制文件2进行解压：



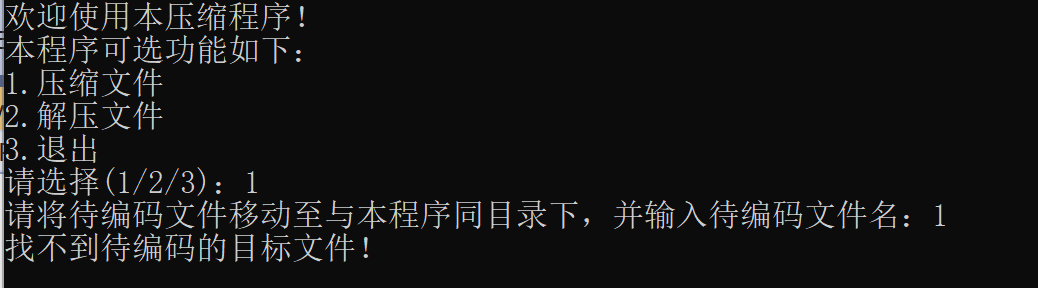
得到解压文件3.txt为：



结束运行:



错误数据输出结果范例:



1. **概要设计**

**算法思路概述：**

本程序主要实现功能：

首先根据选择需要实现的功能，

输入’1’则实现压缩功能，输入待压缩文件名并命名压缩处理后的文件名。压缩步骤如下：读入待压缩文件（每次读入固定字节数，多次读入，根据提前统计的文件字符数作为判断读入是否停止的标准），计算每个字符出现次数，以此作为权重构造哈夫曼树的叶子结点（，再依此构建出完整的哈夫曼树（每次寻找权值最小的无双亲的两个结点作为左右孩子生成一个新结点，结点数据置为256表示无法取到）。遍历哈夫曼树的叶子结点得到一个字符和其对应的哈夫曼编码的匹配（根节点到叶子结点的路径按左0右1得到一串01字符串，即为叶子结点存的字符对应的哈夫曼编码）。将哈夫曼树存入输出文件后，将输入文件的字符转换为对应的哈夫曼编码产生一串01字符串，每8位转换为1个字符生成新ascii码字符串（这个串可能很长，所以每处理一次读入的固定字符数的待压缩文件就将处理后的字符串输出到压缩文件里），存入输入文件，至此压缩完成。

输入’2’则实现解压功能，输入待解压缩文件名并命名解压缩处理后的文件名。首先读入我们存入的哈夫曼树，然后读入压缩的哈夫曼编码对应字符串（每次读入固定字节数，多次读入，根据提前统计的哈夫曼编码对应字符流的字符数作为判断读入是否停止的标准）。根据哈夫曼编码是非前缀编码，我们从哈夫曼树的根节点开始，对每次读入字符流的一个字符，然后根据0/1选择进入左子树/右子树，如果遇到叶子结点就取其存储的字符，然后自下一个字符起重新从哈夫曼树的根节点开始进行上述操作得到新的字符。如此我们得到一个ascii字符串，这就是解压的文件内容，每处理完一次读入的部分压缩文件后就输出解压内容到解压文件中。

输入’3’则实现退出程序的功能，程序停止。在输入’3’之前程序可以多次执行不同压缩/解压功能。

输入其他内容则会有报错提醒，并且可以重新选择功能。

**数据结构类型的定义：**

// 哈夫曼树结点

struct HuffmanNode

{

char data; // 每八位二进制视作一个字符读入，最后不足八位的再另外视为一个字符

long weight;

int parent, lchild, rchild;

};

// 哈夫曼树

struct HuffmanTree

{

HuffmanNode node[TREESIZE - 1];

int len;

};

// 数据（单字符对应的数值）和哈夫曼编码的匹配结点

struct MapNode

{

char data;

char Bin[CODESIZE + 1]; // 存二进制串

};

// 基于哈夫曼树得到的数据和哈夫曼编码的一整组匹配

struct Map

{

MapNode node[LEAFSIZE];

int len;

};

**主程序流程：**

HuffmanCodeMap.len <- 0

OK <- true // 判断是否正确读入文件，若未能正确读入则直接结束程序

memset(TempCode, 0, sizeof(TempCode))

printf("欢迎使用本压缩程序！")

while (1)

printf("\n本程序可选功能如下：\n1.压缩文件\n2.解压文件\n3.退出\n请选择(1/2/3)：")

choice <- getchar()

ClearBuffer()

if (choice == '1')

then

printf("请将待编码文件移动至与本程序同目录下，并输入待编码文件名：")

scanf("%s", sourcefile)

getchar()

OK <- Init(sourcefile)

if (!OK) continue

CreatHaffmanTree()

CreatHaffmanCodeMap(H.len - 1, TempCode, 0)

printf("请命名压缩文件（不超过%d个字符）：", NAMESIZE)

scanf("%s", encodefile)

getchar()

OK <- Encode(sourcefile, encodefile)

if (!OK) then continue

PrintMessage()

else if (choice == '2')

then

printf("请将待编码文件移动至与本程序同目录下，并输入待编码文件名：")

scanf("%s", sourcefile)

getchar()

printf("请命名解压文件（不超过%d个字符）：", NAMESIZE)

scanf("%s", decodefile)

getchar()

OK <- Decode(sourcefile, decodefile)

if (!OK) then continue

else if (choice == '3') then break

else then printf("输入错误，无法实现任何功能！\n")

printf("\n成功退出！欢迎下次使用！")

return 0

end

**各层次模块层次调用关系：**

通过输入值来判断如何执行程序，若输入为3程序直接结束，若输入的是1，则main模块调用ClearBuffer来清理缓存区，调用Init模块来读入源文件保存的结点，调用CreatHaffmanCodeMap模块来创建哈夫曼树，在CreatHaffmanCodeMap中调用SearchNode模块来寻找结点中的自由的两个最小结点，调用CreatHaffmanCodeMap模块,通过迭代来寻找每一个结点对应的编码（左子树为0，右子树为1），调用Encode模块来压缩文件，调用PrintMessage模块来打印哈夫曼树的结点和哈夫曼树的编码结点。如果输入值为2，调用Decode模块来解压文件。

1. **详细设计**

**各种函数的设计：（伪代码）**

1. 初始化哈夫曼树结点：

H.len <- 0

fptr <- fopen(filename, "rb")

if fptr == NULL

printf("找不到待编码的目标文件！\n")

return false

fseek(fptr, 0, SEEK\_END)

count <- ftell(fptr)

fseek(fptr, 0, SEEK\_SET)

while count

int size <- count >= READSIZE ? READSIZE : count

fread(source, sizeof(char), size, fptr)

for int i <- 0; i < size; i++

find <- false

for j <- 0; j < H.len; j++

if H.node[j].data == source[i]

H.node[j].weight++

find <- true

break

end

if !find

H.node[H.len].data <- source[i]

H.node[H.len].weight <- 1

H.node[H.len].lchild <- H.node[H.len].rchild <- H.node[H.len].parent <--1;

H.len++

end

end

count <- count - size

end

fclose(fptr)

return true;

1. 构建完整的哈夫曼树：

count <- H.len - 1; // 哈夫曼树只有度为2/0的结点，num(d=2) = num(d=0) - 1

while count--

pos1, pos2;

SearchNode(H.len, pos1, pos2)

H.node[H.len].lchild <- pos1, H.node[H.len].rchild <- pos2, H.node[H.len].parent <- -1;

H.node[H.len].weight <- H.node[pos1].weight + H.node[pos2].weight

H.node[pos1].parent <- H.node[pos2].parent = H.len

H.len++

end

1. 构建哈夫曼树中寻找已有结点中权值最小的两个结点

pos1 <- -1, pos2 <-= -1; //权值上pos1为最小,pos2第二小

for i <- 0; i < Range; i++

// 有双亲的结点不能作为根节点被合并

if H.node[i].parent != -1

continue

// 新结点权值比已有的两个小

if pos1 == -1 || H.node[pos1].weight > H.node[i].weight

pos2 = pos1, pos1 = i

// 新结点权值在两个结点的权值之间

else if pos2 == -1 || (H.node[pos1].weight < H.node[i].weight && H.node[i].weight < H.node[pos2].weight

pos2 = i

end

return;

1. 得到哈夫曼编码匹配：

if H.node[root].lchild == -1 && H.node[root].rchild == -1)

HuffmanCodeMap.node[HuffmanCodeMap.len].data <- H.node[root].data

strcpy(HuffmanCodeMap.node[HuffmanCodeMap.len].Bin, TempCode)

HuffmanCodeMap.len++

end

if H.node[root].lchild != -1

TempCode[len] <- '0'

CreatHaffmanCodeMap(H.node[root].lchild, TempCode, len + 1)

TempCode[len] <- 0

end

if H.node[root].rchild != -1

TempCode[len] <- '1'

CreatHaffmanCodeMap(H.node[root].rchild, TempCode, len + 1)

TempCode[len] <- 0

end

return

1. 编码：

fptrS = fopen(sourcefile, "rb")

fptrD = fopen(destfile, "wb")

if (fptrS == NULL || fptrD == NULL)

printf("文件名出现错误，编码失败！\n")

if (fptrS)

fclose(fptrS)

if (fptrD)

fclose(fptrD)

return false

end

memset(source, 0, sizeof(source))

memset(temp, 0, sizeof(temp))

memset(dest, 0, sizeof(dest))

fseek(fptrS, 0, SEEK\_END)

count = ftell(fptrS) // 除去EOF的所有字符

fseek(fptrS, 0, SEEK\_SET)

fwrite(&H, sizeof(HuffmanTree), 1, fptrD)

while (count > 0)

int size <- count > READSIZE ? READSIZE : count

fread(source, sizeof(char), size, fptrS)

for (int i <- 0; i < size; i++)

ch <- source[i]

for (int j <- 0; j < HuffmanCodeMap.len; j++)

if (HuffmanCodeMap.node[j].data == ch)

strcat(temp, HuffmanCodeMap.node[j].Bin)

break

end

end

templen <- strlen(temp), destlen <- 0

rev <- templen % 8

for int i <- 0; i < templen / 8; i++

int ch <- 0

for int j <-0; j < 8; j++ // 低地址高位的二进制码==>0-255==>字符

if j

ch <- ch \* 2 + (temp[8 \* i + j] - '0')

else

ch <- ch \* 2 - (temp[8 \* i + j] - '0')

dest[destlen++] <- (char)ch

end

fwrite(dest, sizeof(char), destlen, fptrD)

count <- count - size // 不足八位的留着下次循环补齐

for int i <- 0; i < rev; i++

copy[i] <- temp[templen - rev + i];

memset(temp, 0, sizeof(temp))

for int i <- 0; i < rev; i++

temp[i] <- copy[i];

memset(source, 0, sizeof(source))

memset(dest, 0, sizeof(dest))

end

rev<- strlen(temp)

if rev

int ch <- 0

for i <- rev; i < 8; i++ // 后面应该补8-rev个'0'

temp[i] <- '0'

for i <- 0; i < 8; i++

if i

ch <- ch \* 2 + (temp[i] - '0')

else

ch <- ch \* 2 - (temp[i] - '0')

dest[0] <- (char)ch

fwrite(dest, sizeof(char), 1, fptrD)

end

fputc((char)rev, fptrD) // 存储最后数位剩余的信息，多占一个字节

printf("压缩成功！压缩文件为%s\n", destfile)

fclose(fptrS)

fclose(fptrD)

return true

1. 解码：

fptrS <- fopen(sourcefile, "rb")

fptrD <- fopen(destfile, "wb")

if fptrS == NULL || fptrD == NULL

printf("文件名出现错误，解码失败！\n")

if fptrS

fclose(fptrS)

if fptrD

fclose(fptrD)

return false

end

memset(source, 0, sizeof(source))

memset(temp, 0, sizeof(temp))

memset(dest, 0, sizeof(dest))

fseek(fptrS, -1, SEEK\_END); // 定位到辅助信息处，倒数顺序：EOF 辅助信息 存储的最后一位(可能只有部分有用)

count <- ftell(fptrS); // count为有用信息的字节数(最后几个bit可能为无效信息)

rev = (int)fgetc(fptrS) // 最后一个字节的有效比特位数

fseek(fptrS, 0, SEEK\_SET)

fread(&H, sizeof(HuffmanTree), 1, fptrS)

count <- count - ftell(fptrS)

while (count) // 若feof(fptr)则count=0

size = count > READSIZE ? READSIZE : count

fread(source, sizeof(char), size, fptrS)

for i <- 0; i < size; i++

int ch <-= (unsigned char)source[i]

memset(copy, 0, sizeof copy)

for j <- 7; j >= 0; j-- //最先输出低位,低位在高地址

copy[j] <- '0' + (ch % 2)

ch <- ch / 2

end

strcat(temp, copy)

templen = strlen(temp), destlen = 0;

index = 0;

while templen - index >= CODESIZE

root = H.len - 1;

while H.node[root].lchild != -1 && H.node[root].rchild != -1

if temp[index] == '0'

root <- H.node[root].lchild

else if (temp[index] == '1')

root <-H.node[root].rchild

index++

end

dest[destlen++] <- H.node[root].data

end

fwrite(dest, sizeof(char), destlen, fptrD)

count <- count- size

memset(copy, 0, sizeof(copy))

for i <- 0; i < templen - index; i++ // 不足CODESIZE位的留着下次循环再找

copy[i] <- temp[index + i]

memset(temp, 0, sizeof(temp))

for i <- 0; i < templen - index; i++

temp[i] <-= copy[i]

memset(source, 0, sizeof(source))

memset(dest, 0, sizeof(dest))

end

templen <- strlen(temp) - (8 - rev) // 8-rev位是人工添加的‘0’，是无效信息

index <- 0, destlen <- 0;

while index < templen

root <- H.len - 1;

while (H.node[root].lchild != -1 && H.node[root].rchild != -1)

if (temp[index] == '0')

root <-H.node[root].lchild;

else if (temp[index] == '1')

root <- H.node[root].rchild;

index++

end

dest[destlen++] <- H.node[root].data

end

fwrite(dest, sizeof(char), destlen, fptrD)

printf("解压成功！解压文件为%s\n", destfile)

fclose(fptrS)

fclose(fptrD)

return true

1. 输出哈夫曼树和哈夫曼编码匹配的信息:

printf("构造的哈夫曼树有%d个结点，其中叶子结点有%d个，如下所示：\n", H.len, (H.len + 1) / 2)

printf("node\t\tdata\t\tweight\t\tlchild\t\trchild\t\tparent\n")

for i = 0; i < H.len; i++

printf("%d\t\t%d\t\t%ld\t\t%d\t\t%d\t\t%d\n", i, i <= (H.len / 2) ? H.node[i].data : 256, H.node[i].weight, H.node[i].lchild, H.node[i].rchild, H.node[i].parent)

printf("生成的哈夫曼编码如下：\n")

printf("data\t\tcode\n")

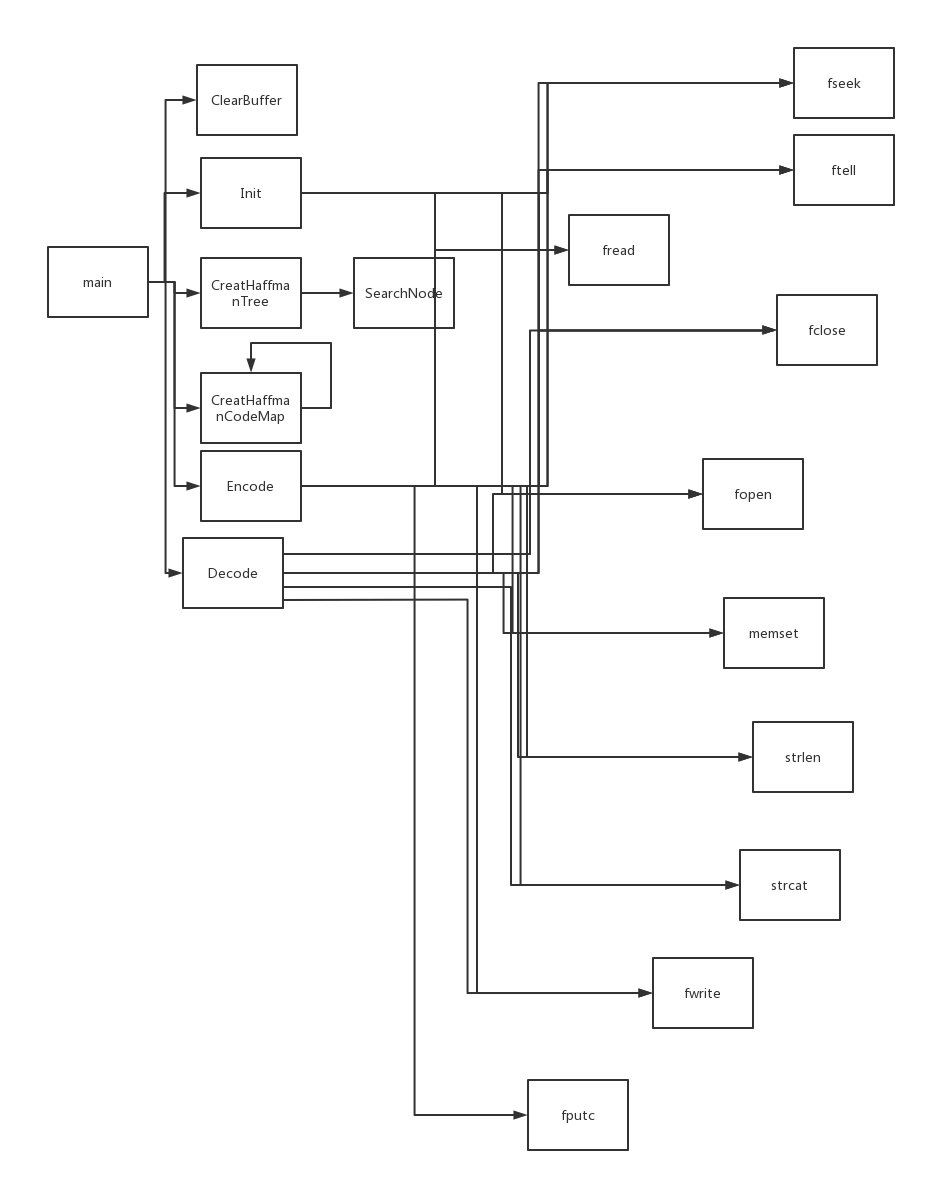
for i = 0; i < HuffmanCodeMap.len; i++)

printf("%d\t\t%s\n", HuffmanCodeMap.node[i].data, HuffmanCodeMap.node[i].Bin)

return;

end

**函数调用关系图：**

****

1. **调试分析报告**

**调试过程中遇到的问题：**

首先，读写文件的时候一定要注意判断文件类别，错误的读入模式会导致不正常的读写。其次，我们注意到压缩文件的时候得到的01字符流不一定刚好能补齐八位，因此最后我们补一定数量的‘0’凑齐八位转换成字符，并且我们要把补充的‘0’的个数作为辅助信息也写入压缩文件内才能保证后续的解压过程正常进行。最后，通过查阅资料我们发现.jpg，.jpeg等格式的文件已经是被压缩过的了，因此通过本程序进行压缩所实现的压缩效果并不明显。

**算法时空分析：**

设输入文件大小为n  
Init o(n)  
CreateHuffmanTree o(1)  
 //新构建的非叶子结点最多有255个  
SearchNode o(1) //结点最多有511个，树的高度最大也只能是256，搜索次数也为常数级  
CreatHaffmanCodeMap o(1)   
Encode o(n)  
Decode o(n)  
PrintMessage o(1)  
ClearBuffer o(m) //m为用户在命令行键入的字符数  
main o(n)

**改进设想和经验体会**：

读写的时候，可以考虑分块读入，在保证内存开销较小的同时提升运行速度。算法的思路比较简单，读入字符统计字符数作为权值，构建哈夫曼树得到对应编码，然后编码就是把每个字符转换为01字符串然后再按二进制的方式分块写入压缩文件，解码就是二进制读入压缩文件，通过哈夫曼树找到编码对应结点上的字符，把得到的字符再分块写入解压文件。

1. **用户使用说明**

步骤1：

用户打开程序，首先选择输入 文件路径和文件名。

步骤2：

根据文件类型选择功能1，2 ，3

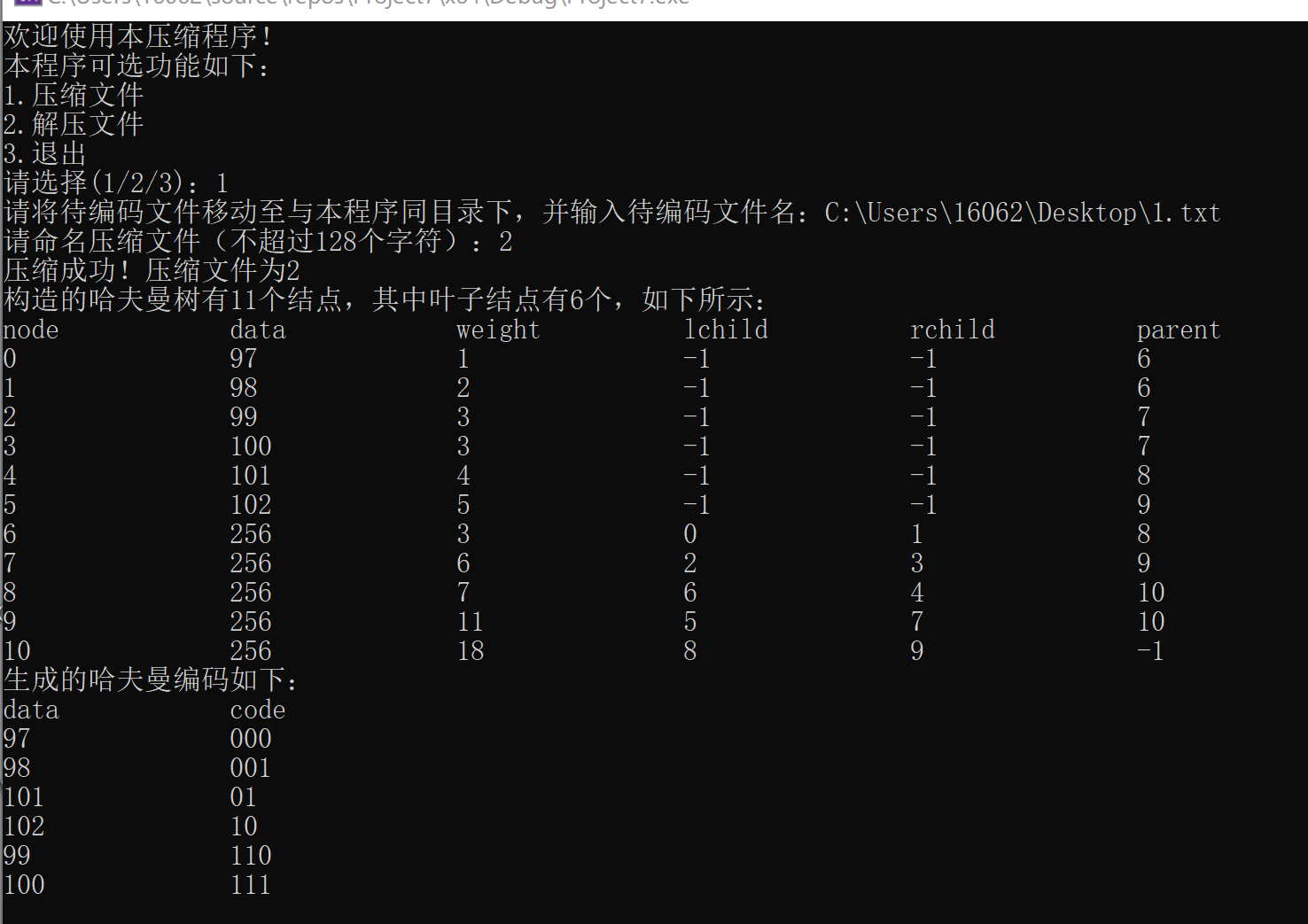
步骤3：

输入文件路径和文件名，如果是功能1，2，需要输入压缩后或解压后的文件名

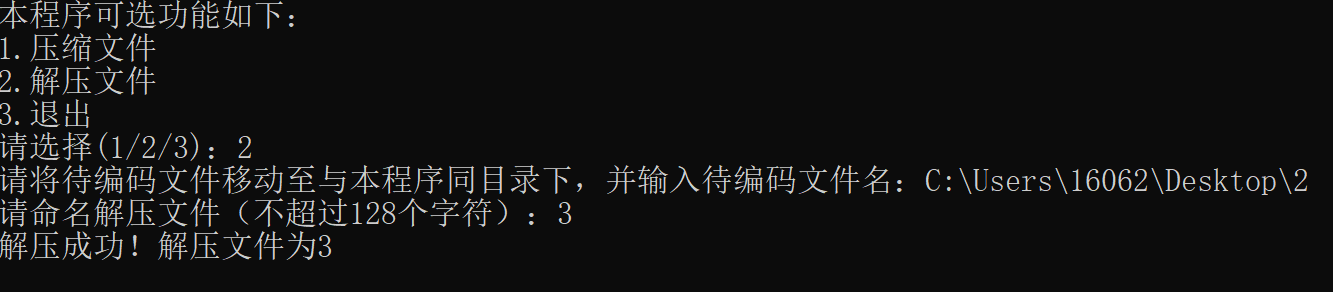
步骤4：

输入解压文件名

1. **测试结果**

1.选择功能1时候：  


1. 选择功能2：



1. 选择功能3：

