实验 3: 用huffman压缩技术实现对任意文件的压缩和解压缩处理

1. 问题描述

要求对所有的文件类型进行压缩，压缩之后的文件后缀名为huff。同时，可以对所有后缀名为huff的压缩文件进行解压缩。

1. 题目2中以1个字节(8bit)为单位进行huffman编码
2. 对任意文件进行压缩后可以输出一个后缀名为huff的单文件，并且可以对任意一个后缀名为huff的单文件进行解压还原出原文件。
3. 群内将提供10不同种类的文件包括文档、图片、视频、可执行文件等进行压缩测试，要求可以完成压缩和解压的步骤，并且解压出来的文件没有任何损失。

输入要求: 假定文件名为 `huff`, 则若 `argc` 为2， 即 `./huff xxx` 则为 压缩，若 `argc` 为 3， 即 `./huff xxx.huff xxx`, 即为 解压缩

2. 算法的描述

2.1 数据结构的描述

a. 压缩 压缩出的文件名为 输入的文件名`.huff`, 使用一个 Map 对 0-255 的 ascii 码进行处理:

```
struct OriMapNode map[MAP_SIZE];
```

使用 `qsort()` 对该 `map` 进行排序后，使用 `BuildTree` 建立一个 Huffman Tree，然后建立新的 `map` 数组 `CodeMap`, 它的每个元素都是 `struct MapNode`, 包含了 `code` 和 `code` 的长度 `length`. 调用 `GenCode()` 对 `CodeMap` 里的每个元素生成对应的哈夫曼编码。HTree 顶部的 `weight` 即为整个 Huffman Tree 的总长度。我们使用一个头(Header) 存储这些信息，然后对每个源文件中的字节，在 CodeMap 中找到相应的编码，并写入文件中，完成写入。

b. 解压缩 程序起先读取文件中预置的头(Header), 然后使用 header 中自带的 map 重建 Huffman Tree，然后对每个 bit，在 Huffman Tree 中进行查找，最后找到相应的原文件的 byte，并写入原文件。

2.2 程序结构的描述

`main.c` 中主要的函数有:

- `main()` 处理输入，确定是解压缩还是压缩
- `compress()` 压缩文件
- `depress()` 解压缩文件
- `my_cmp()` `qsort()` 的比较函数
- `BuildTree()` 建立 Huffman Tree
- `GenCode()` 生成 Huffman Code
- `__gen_code()` 由 `GenCode()` 调用，递归地生成代码

3. 调试分析

本程序的调试主要通过手写和使用 hex editor 来查看压缩完成的文件来判断算法是否正确

4. 实验体会和收获

这是我第一次写使用位操作来 1 个 bit 1 个 bit 处理文件的程序，从原来的不熟悉到后来的越写越顺，进一步深化了对 HUffman Tree 的认识。