**文件压缩小程序**

**课程名称：面向对象程序设计**

**姓名：张宇**

**院系：电子与信息工程学院计算机科学与技术专业**

**班级：计算机科学与技术1班**

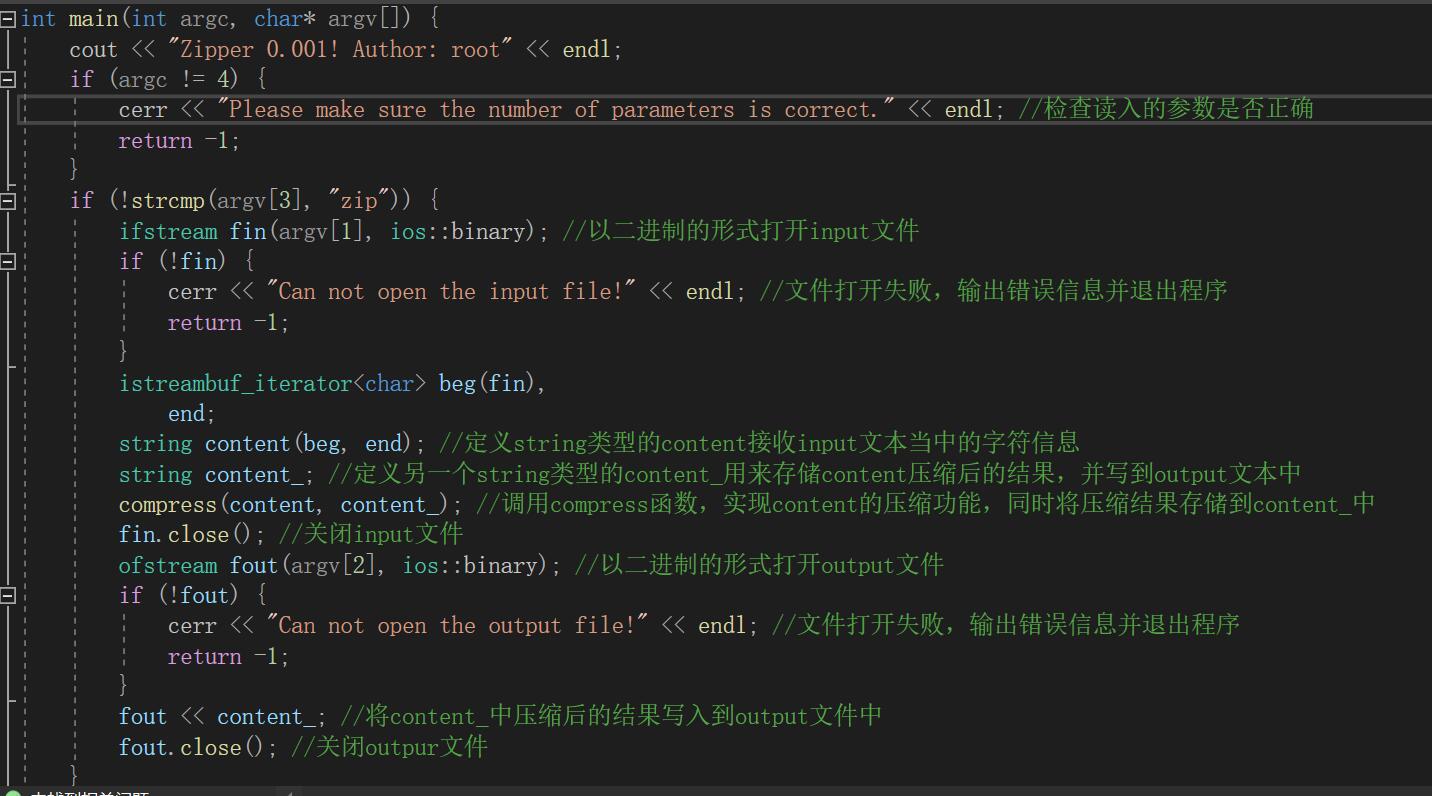
**学号：2251745**

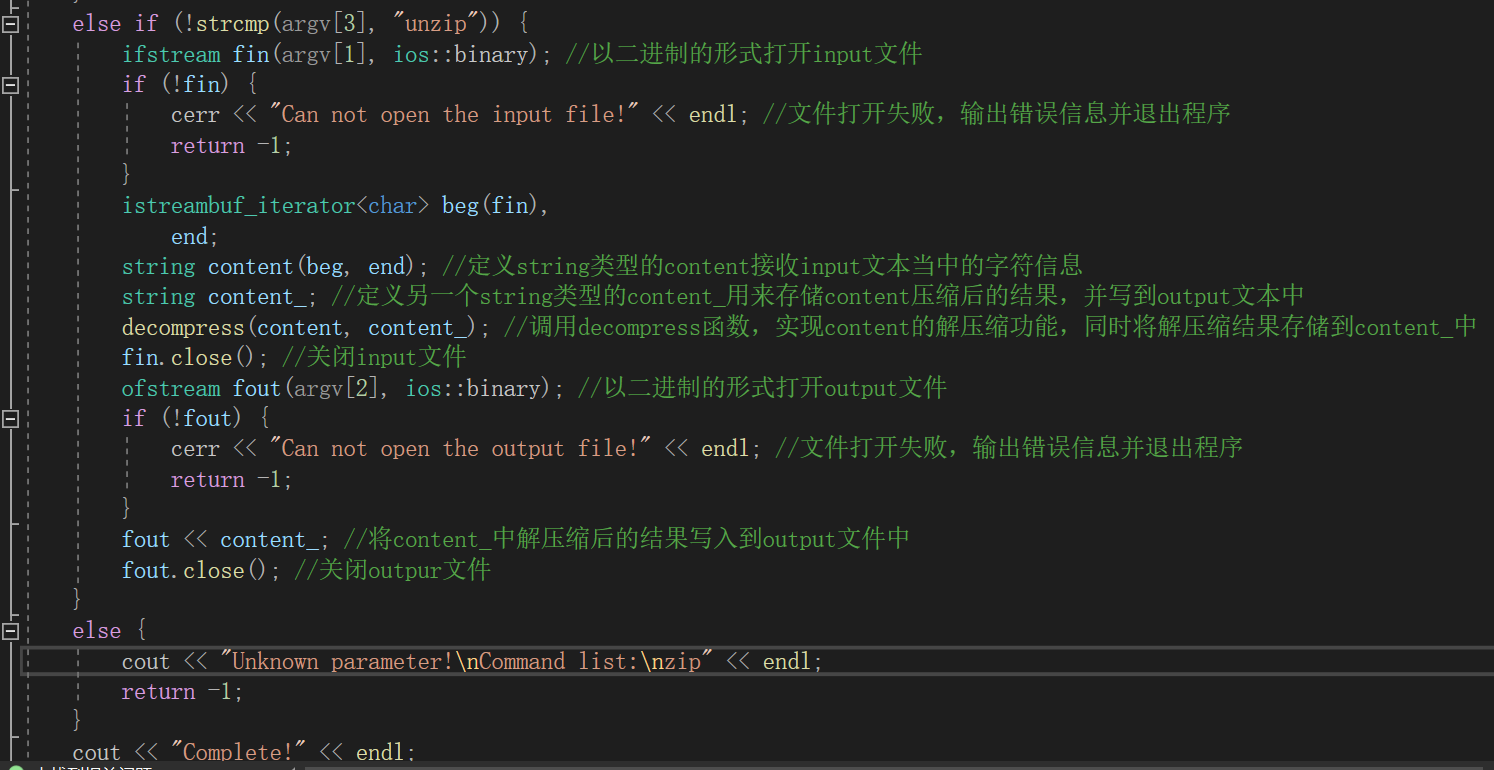
**完成日期：2023年11月21日**

**1、设计思路与功能描述**

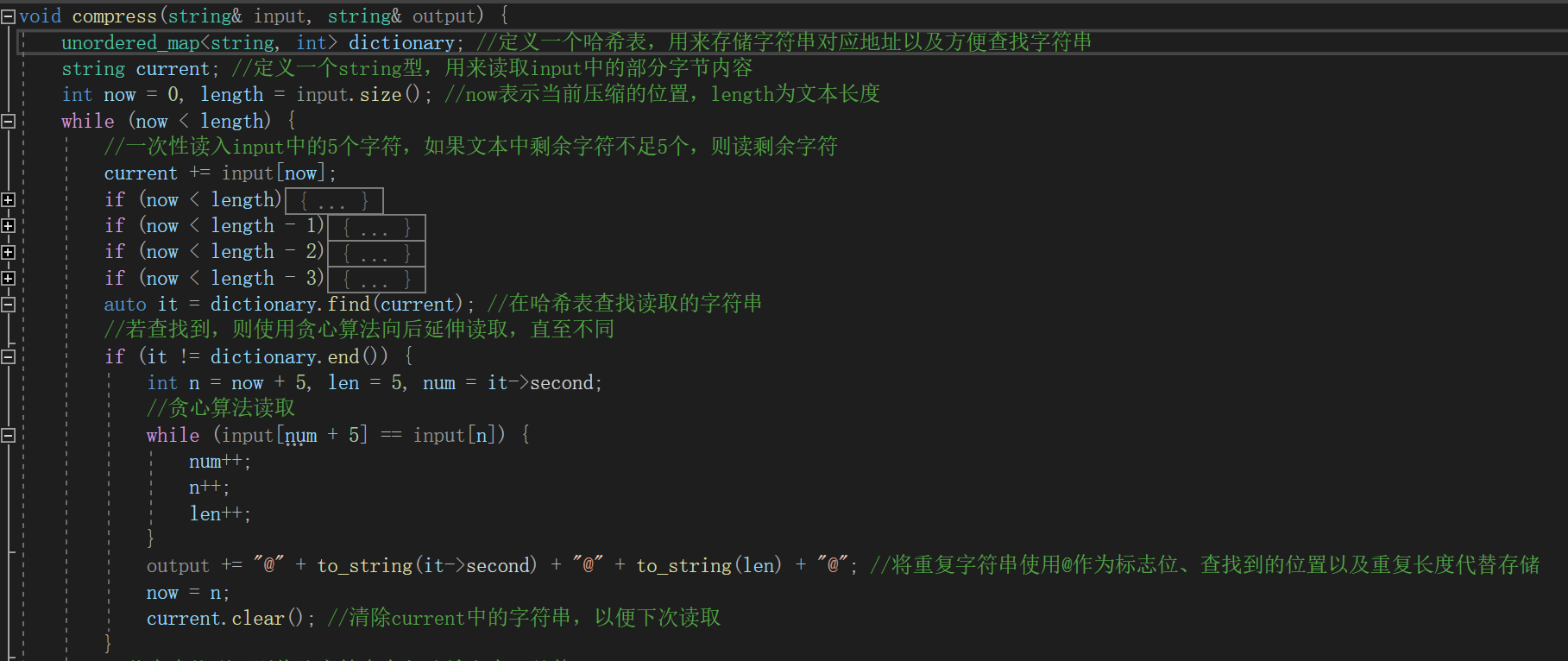
在刚拿到这个题目的时候还是处于一个比较懵的状态，一直都没有什么写的思路，然后就想着先看看参考资料里面都有些什么，就看到了学长们的方法，面对哈夫曼树、Lz77算法和Lz4算法，也是毫无疑问就选择了最优化的Lz4算法，然而其难度也有点出乎意料了。首先是对于哈希表、动态缓冲区等不熟悉的名词，又要花时间去做了解，才可能勉强看懂；其次是参考资料分享 了很多网站之类的可以用作学习，里面的内容也参差不齐，有很多看了以后感觉更加一头雾水的，导致花费了大量的时间去查阅资料，而代码是一个字也没敲，反而是越看越觉得难，等好不容易开始写了才发现已经过去了两天时间；正在上手写的时候才发现实操是真的不一样，又在写的时候边写边学，这也导致了其实对于算法并不满意，有一些更值得优化的地方，但是涉及到哈希表的知识，实在是没搞明白。

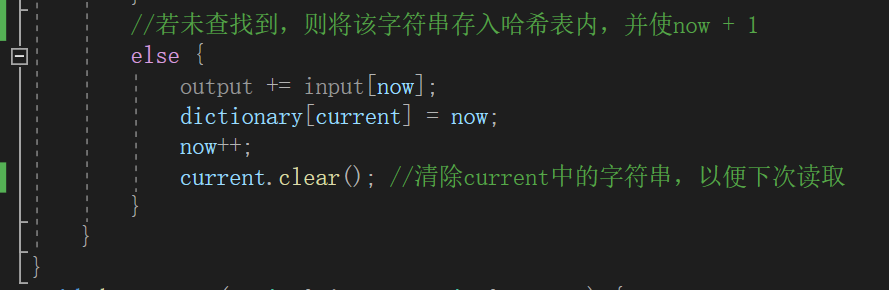
首先是main函数，main函数主要是用于传入参数的数量判断、输入文件和输出文件的打开、读取、写入和关闭等，然后是根据参数选择调用compress函数还是decompress函数，实现压缩和解压功能。



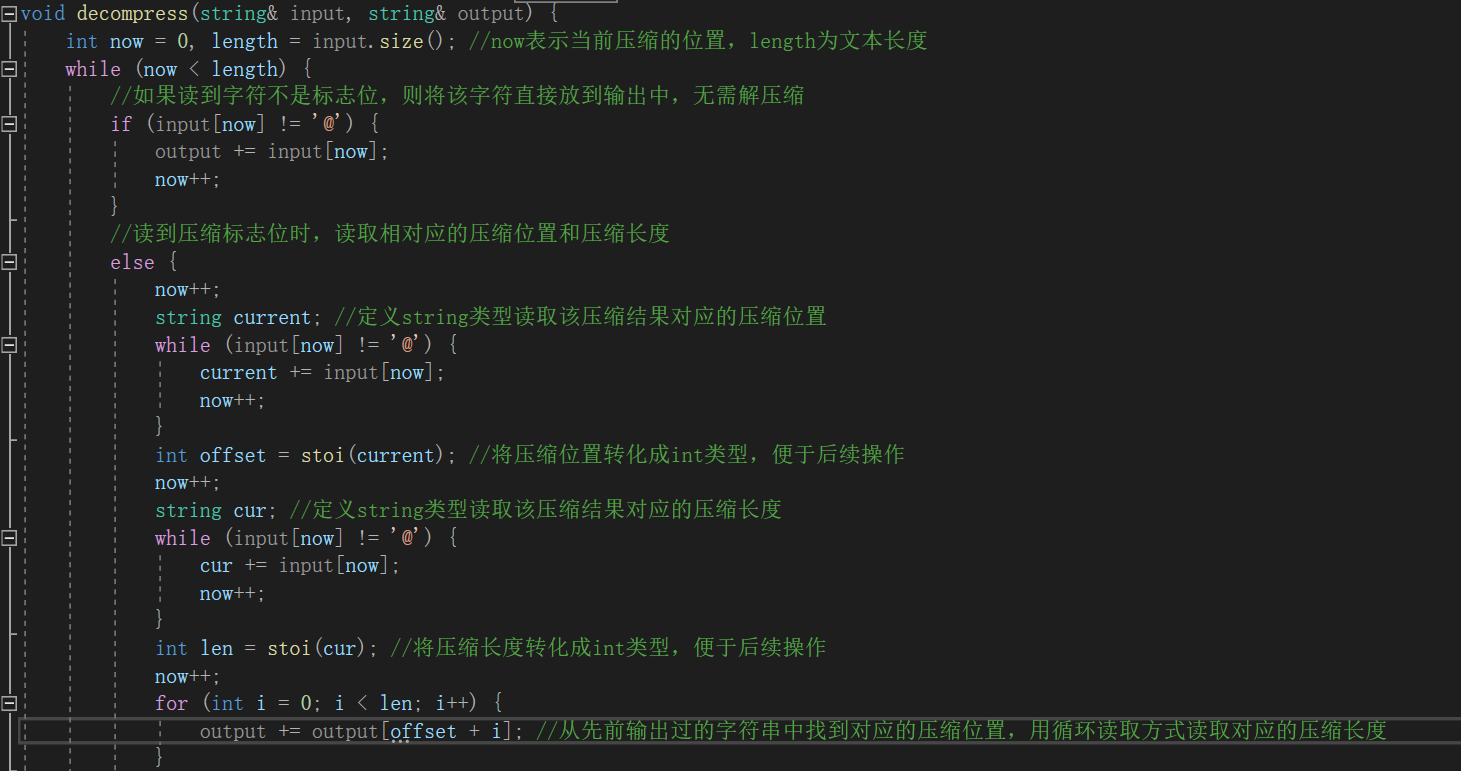


然后是compress函数，用于文件的压缩，传入从input文件中读取的字符串和待写入到output文件中的字符串为参数进行压缩。其中使用了哈希表以及贪心算法等知识，压缩时间约为2s左右，压缩率在40%。





最后是decompress函数，用于文件的解压缩，传入从input文件中读取的字符串和待写入到output文件中的字符串为参数进行解压缩。按照压缩文件的算法实现相对应的解压缩功能，遇到标志位时进行相应操作以实现功能，解压缩时间在500ms左右。



**2、在实验过程中遇到的问题与解决方法**

（1）问题：压缩时一开始以3个字符为标准，每次读取3个字符，该字符串若能在哈希表中查找到，则再向后读3个字符，直至在哈希表中查找不到。这种做法导致压缩率一般，只能压缩以3为倍数的重复字符串，效率一般。

解决方法：重新查阅资料后看到了贪心算法，即当在哈希表中查找到当前字符串后，就将当前字符串和哈希表中存储的字符串同时向后延伸，直至不是相同字符为止。贪心算法极大保证了压缩的效率，更具有一般性和普适性。

（2）问题：在解压缩时，对于未加标志位的压缩结果无法区分出相对应的压缩位置和压缩长度。

解决方法：在压缩时添加压缩标志位来区分出压缩位置和压缩长度，这就需要选择合适的压缩标志位，这本题中我选择了@字符，其实仅仅是因为在压缩结果查找中发现没有这一字符，因此可作为标志位，但显然这并不具有普适性，对于别的文本可能就无法进行正确的压缩。

（3）问题：在解压缩时，一开始对于读取到的压缩位置和压缩长度，是在input文本中找到对应的位置开始循环读取，但总出现越界的情况。

解决方法：在调试时对比解压后的文本与原文本时发现，应该在output文本中寻找相对应的位置开始循环读取压缩长度，就可以解决循环读取文本时出现越界的情况。

（4）问题：在压缩时发现当设置每次读取的字符串长度改变时，压缩率也会改变，也就是当每次一同读取的字符串越长，最终的压缩率也会越低。

解决方法：这涉及到在哈希表存储的字符长度，即哈希表更深层次的知识，对哈希表的理解不够透彻，没能有很好的解决方法。

**3、心得体会**

本次作业实现的文件压缩功能所使用到的基本都是没有学过的知识，例如哈希表、贪心算法等等，这就需要我们各种途径查找相关的资料，学习相关的知识，并且能够在理解掌握之后在自己的代码中实现相应的功能，这就很考验我们的理解和学习能力了，同时这也是最需要时间的工作。因此，如何快速、高效地寻找到合适的资料学习，就很需要我们的检索能力了。我也是在这上面消耗了大量的时间，但其实效果也并不是很好，只能说是越看感觉自己越乱，所以还需要值得注意的是在感觉自己理解掌握了一点后，就应该及时地敲敲代码，自己动手实践一下，“纸上得来终觉浅，须知此事要躬行”，只有当自己敲代码的时候才知道到底哪里掌握得还不够好，也才可以相对应地去学习。

写了很多次大作业后感觉到其实每次大作业看着都很唬人，尤其是没有循序渐进的过程时，总觉得自己好像做不出来，掌握的知识不够解决问题，但其实只有自己动起手来，一点一点地完成代码，才能知道自己的不足，也才有学习的方向和动力，一个个小功能的实现，最终叠加起来，原来一个看似不可能的大作业也就一步步的解决了。

**4、源代码**

#include <fstream>

#include <iostream>

#include <string>

#include <cstring>

#include <unordered\_map>

using namespace std;

void compress(string& input, string& output) {

unordered\_map<string, int> dictionary; //定义一个哈希表，用来存储字符串对应地址以及方便查找字符串

string current; //定义一个string型，用来读取input中的部分字节内容

int now = 0, length = input.size(); //now表示当前压缩的位置，length为文本长度

while (now < length) {

//一次性读入input中的5个字符，如果文本中剩余字符不足5个，则读剩余字符

current += input[now];

if (now < length) {

current += input[now + 1];

}

if (now < length - 1) {

current += input[now + 2];

}

if (now < length - 2) {

current += input[now + 3];

}

if (now < length - 3) {

current += input[now + 4];

}

auto it = dictionary.find(current); //在哈希表查找读取的字符串

//若查找到，则使用贪心算法向后延伸读取，直至不同

if (it != dictionary.end()) {

int n = now + 5, len = 5, num = it->second;

//贪心算法读取

while (input[num + 5] == input[n]) {

num++;

n++;

len++;

}

output += "@" + to\_string(it->second) + "@" + to\_string(len) + "@"; //将重复字符串使用@作为标志位、查找到的位置以及重复长度代替存储

now = n;

current.clear(); //清除current中的字符串，以便下次读取

}

//若未查找到，则将该字符串存入哈希表内，并使now + 1

else {

output += input[now];

dictionary[current] = now;

now++;

current.clear(); //清除current中的字符串，以便下次读取

}

}

}

void decompress(string& input, string& output) {

int now = 0, length = input.size(); //now表示当前压缩的位置，length为文本长度

while (now < length) {

//如果读到字符不是标志位，则将该字符直接放到输出中，无需解压缩

if (input[now] != '@') {

output += input[now];

now++;

}

//读到压缩标志位时，读取相对应的压缩位置和压缩长度

else {

now++;

string current; //定义string类型读取该压缩结果对应的压缩位置

while (input[now] != '@') {

current += input[now];

now++;

}

int offset = stoi(current); //将压缩位置转化成int类型，便于后续操作

now++;

string cur; //定义string类型读取该压缩结果对应的压缩长度

while (input[now] != '@') {

cur += input[now];

now++;

}

int len = stoi(cur); //将压缩长度转化成int类型，便于后续操作

now++;

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

output += output[offset + i]; //从先前输出过的字符串中找到对应的压缩位置，用循环读取方式读取对应的压缩长度

}

}

}

}

int main(int argc, char\* argv[]) {

cout << "Zipper 0.001! Author: root" << endl;

if (argc != 4) {

cerr << "Please make sure the number of parameters is correct." << endl; //检查读入的参数是否正确

return -1;

}

if (!strcmp(argv[3], "zip")) {

ifstream fin(argv[1], ios::binary); //以二进制的形式打开input文件

if (!fin) {

cerr << "Can not open the input file!" << endl; //文件打开失败，输出错误信息并退出程序

return -1;

}

istreambuf\_iterator<char> beg(fin),

end;

string content(beg, end); //定义string类型的content接收input文本当中的字符信息

string content\_; //定义另一个string类型的content\_用来存储content压缩后的结果，并写到output文本中

compress(content, content\_); //调用compress函数，实现content的压缩功能，同时将压缩结果存储到content\_中

fin.close(); //关闭input文件

ofstream fout(argv[2], ios::binary); //以二进制的形式打开output文件

if (!fout) {

cerr << "Can not open the output file!" << endl; //文件打开失败，输出错误信息并退出程序

return -1;

}

fout << content\_; //将content\_中压缩后的结果写入到output文件中

fout.close(); //关闭outpur文件

}

else if (!strcmp(argv[3], "unzip")) {

ifstream fin(argv[1], ios::binary); //以二进制的形式打开input文件

if (!fin) {

cerr << "Can not open the input file!" << endl; //文件打开失败，输出错误信息并退出程序

return -1;

}

istreambuf\_iterator<char> beg(fin),

end;

string content(beg, end); //定义string类型的content接收input文本当中的字符信息

string content\_; //定义另一个string类型的content\_用来存储content压缩后的结果，并写到output文本中

decompress(content, content\_); //调用decompress函数，实现content的解压缩功能，同时将解压缩结果存储到content\_中

fin.close(); //关闭input文件

ofstream fout(argv[2], ios::binary); //以二进制的形式打开output文件

if (!fout) {

cerr << "Can not open the output file!" << endl; //文件打开失败，输出错误信息并退出程序

return -1;

}

fout << content\_; //将content\_中解压缩后的结果写入到output文件中

fout.close(); //关闭outpur文件

}

else {

cout << "Unknown parameter!\nCommand list:\nzip" << endl;

return -1;

}

cout << "Complete!" << endl;

return 0;

}