МИНЕСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧЕРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«Брестский государственный технический университет»

Кафедра «Интеллектуальные информационные технологии»

Лабораторная работа №3

По дисциплине «Основы информационных технологий»

За 5 семестр

Тема: «Сжатие (архивация) и восстановление (разархивация) файлов»

Выполнила:

студентка 3 курса

группы АС-56

Карпенко М.В.

Проверил:

Пролиско Е.Е.

Брест 2021

***Задание***

Используя коды сжатия, полученные в предыдущей лабораторной работе, построить алгоритмы сжатия (архивация) файлов. Заархивировать три файла, степень сжатия которых, с вашей точки зрения, может быть максимальной. Обосновать этот выбор. Создать программу восстановления (разархивации), которая, используя имеющийся файл с кодировочной таблицей и ваш архив, восстанавливает исходный файл.

***Вариант 44***

*Код программы:*

#include <iostream>  
#include <cmath>  
#include <fstream>  
#include <bitset>  
#include <string>  
#include <sstream>  
using namespace std;  
const int background = system("color f4");  
struct map {  
int simbol;  
string kod;  
};  
void zip(string filename, string method) {  
//Запись номеров символов и их кодов в словарь  
ifstream tabl;  
tabl.open(filename + method, ios::in | ios::binary);  
char stroka1[50];  
for (int i = 0; i < 3; ++i) tabl.getline(stroka1, 50);  
map dictionary[256];  
string stroka[256];  
for (int i = 0; i < 256; ++i) {  
getline(tabl, stroka[i]);  
istringstream ss(stroka[i]);  
ss >> dictionary[i].simbol;  
ss >> dictionary[i].kod;  
}  
tabl.close();  
for (int i = 0; i < 255; ++i) {//сортируем словарь по номерам символов  
int temp1; string temp2;  
for (int j = 0; j < 255 - i; ++j) {  
if (dictionary[j].simbol > dictionary[j + 1].simbol) {  
temp1 = dictionary[j].simbol;  
temp2 = dictionary[j].kod;  
dictionary[j].simbol = dictionary[j + 1].simbol;  
dictionary[j].kod = dictionary[j + 1].kod;  
dictionary[j + 1].simbol = temp1;  
dictionary[j + 1].kod = temp2;  
}  
}  
}  
ifstream file(filename, ios::in | ios::binary);  
if (!file.is\_open()) {  
cout << "Ошибка считывания файла" << endl;  
return;  
}  
//вес файла  
file.seekg(0, file.end);  
long double size = file.tellg();  
file.seekg(0, file.beg);  
//считывание в массив char  
char\* byte\_as\_char = new char[size];  
file.read(byte\_as\_char, size);  
//переписывание в массив int  
int\* byte\_as\_int = new int[size];  
for (int i = 0; i < size; ++i) {  
byte\_as\_int[i] = byte\_as\_char[i];  
if (byte\_as\_int[i] < 0) {  
byte\_as\_int[i] = byte\_as\_int[i] + 256;  
}  
}  
file.close();  
//запись закодированного сообщения в строку  
string kodirovka;  
for (int i = 0; i < size; ++i) {  
kodirovka += dictionary[byte\_as\_int[i]].kod;  
}  
int subsize;  
int extrasimbols;  
if (kodirovka.length() % 8 == 0) {  
subsize = kodirovka.length() / 8;  
extrasimbols = 8;  
}  
else {  
subsize = 1 + kodirovka.length() / 8;  
extrasimbols = kodirovka.length() % 8;  
}  
string\* subkodirovka = new string[subsize];//разделение на подстроки длиной 8 бит  
for (int i = 0; i < subsize; ++i) {  
subkodirovka[i] = kodirovka.substr(i \* 8, 8);  
}  
bitset<8>\* subbytes = new bitset<8>[subsize];//запись подстрок в массив bitset  
for (int i = 0; i < subsize; ++i) {  
subbytes[i] = bitset<8>{ subkodirovka[i] };  
}  
ofstream archieve;  
archieve.open(filename + ".arh", ios::out | ios::binary);  
for (int i = 0; i < subsize; ++i) {  
archieve << char(subbytes[i].to\_ulong());  
}  
archieve << char(extrasimbols);// записываем последний байт - число символов, которые  
необходимо считать с предпоследнего при разархивировании  
archieve.close();  
cout << endl << "Архивация прошла успешно!" << endl;  
}  
void unzip(string filename, string method) {  
string archievename = filename + ".arh";  
ifstream archieve;  
archieve.open(archievename, ios::in | ios::binary);  
if (!archieve.is\_open()) {  
cout << "Ошибка считывания архива";  
return;  
}  
// определение веса архива  
archieve.seekg(0, archieve.end);  
int size = archieve.tellg();  
archieve.seekg(0, archieve.beg);  
//считывание архива побайтово в массив char  
char\* byte\_as\_char = new char[size];  
archieve.read(byte\_as\_char, size);  
int\* byte\_as\_int = new int[size];  
for (int i = 0; i < size; ++i) {  
byte\_as\_int[i] = byte\_as\_char[i];  
if (byte\_as\_int[i] < 0) {  
byte\_as\_int[i] = byte\_as\_int[i] + 256;  
}  
}  
int extrasimbols = byte\_as\_int[size-1];  
size--;  
string kodirovka;  
bitset<8>\* subbytes = new bitset<8>[size];//запись подстрок в массив bitset  
for (int i = 0; i < size; ++i) {  
subbytes[i] = char(byte\_as\_int[i]);  
if(i<size-1) kodirovka += subbytes[i].to\_string();  
else kodirovka += string(subbytes[i].to\_string()).substr(8-extrasimbols,extrasimbols);  
}  
archieve.close();  
//Запись номеров символов и их кодов в словарь  
ifstream tabl;  
tabl.open(filename + method, ios::in | ios::binary);  
char stroka1[50];  
for (int i = 0; i < 3; ++i) tabl.getline(stroka1, 50);  
map dictionary[256];  
string stroka[256];  
for (int i = 0; i < 256; ++i) {  
getline(tabl, stroka[i]);  
istringstream ss(stroka[i]);  
ss >> dictionary[i].simbol;  
ss >> dictionary[i].kod;  
}  
tabl.close();  
for (int i = 0; i < 255; ++i) {//сортируем словарь по убыванию длины кода  
int temp1; string temp2;  
for (int j = 0; j < 255 - i; ++j) {  
if (string(dictionary[j].kod).length() < string(dictionary[j + 1].kod).length()) {  
temp1 = dictionary[j].simbol;  
temp2 = dictionary[j].kod;  
dictionary[j].simbol = dictionary[j + 1].simbol;  
dictionary[j].kod = dictionary[j + 1].kod;  
dictionary[j + 1].simbol = temp1;  
dictionary[j + 1].kod = temp2;  
}  
}  
}  
ofstream unzipedfile;  
filename = filename.substr(0, filename.rfind('\\')) + "\\!" + filename.substr(filename.rfind('\\') + 1, filename.length());  
unzipedfile.open(filename, ios::out | ios::binary);  
int begsim=0;  
while(begsim < kodirovka.length()){  
for (int j = 0; j < 256; ++j) {  
if (kodirovka.substr(begsim, string(dictionary[j].kod).length()) == dictionary[j].kod) {  
unzipedfile << char(dictionary[j].simbol);  
begsim += string(dictionary[j].kod).length();  
j += 256;  
}  
}  
}  
unzipedfile.close();cout << endl << "Разархивация прошла успешно!" << endl;  
}

int main() {  
setlocale(LC\_ALL, "rus");  
string filename, method;  
int fun;  
cout << "Выберите необходимое действие: \n1)Архивация \n2)Разархивация \n3)Выход \n";  
cin >> fun;  
if (fun == 1) {  
cout << "Введите полный путь к файлу(используя \\\\): ";  
cin >> filename;  
cout << endl << "Введите расширение файла-таблицы: ";  
cin >> method;  
zip(filename, method);  
cout << endl << endl;  
return main();  
}  
if (fun == 2) {  
cout << "Введите полный путь к архиву(используя //): ";  
cin >> filename;  
cout << endl << "Введите расширение файла-таблицы: ";  
cin >> method;  
unzip(filename.substr(0, filename.length() - 4), method);  
cout << endl << endl;  
return main();  
}  
else {  
cout << "Конец выполнения программы\n";  
return 0;  
}  
}

*Результат работы программы:* 





