```
 1: #include <iostream>//библиотека потоков ввода/вывода
 2: #include <vector>//библиотека класса вектор и его методов
 3: #include <map>//библиотека класса таблица и его методов
 4: #include <list>//библиотека класса лист и его методов
 5: #include <fstream>//библиотека файловых потоков
 6: #include <cstring>//библиотека класса строки и их методов
 7:
 8: using namespace std;//объявление стандартнго пространства имен
 9:
10: void encode(void);//функция кодировки
11: void decode(void); //функция декодировки
12: //класс, реализующий узел дерева Хаффмана
13: class Node {
14:     public:
15:      int a;
16:      unsigned char c;
17:      Node *left, *right;
18:      Node() { //конструктор, задающий значения листов по умолчанию
19:         left=right=NULL;
20:      }
21:      Node(Node *L, Node *R) { //конструктор, реализующий сложение частот при
    продвижении к корню
22:       left =  L;
23:       right = R;
24:       a = L->a + R->a;
25:      }
26: };
27: //структрура, содержащая перегрузку оператора ()
28: struct MyCompare {
29:     bool operator()(const Node* l, const Node* r)
30:     const {return l->a < r->a;}
31: };
32:
33: vector<bool> code; //объявление вектора
34: map<unsigned char,vector<bool> > table;// объявление таблицы, используемых
    для представления листа в дереве
35: char name[50], name1[50],name2[50]; //строки, в которые заносятся имена файлов
36: Node* root;//корень дерева Хаффмана
37: int count=0;//счетчик
38: char buf=0;//"символ-буфер"
39:
40: void BuildTable(Node *root) //функция набора префиксного кода для каждого
    листа
41: {
42:   if(root->left!=NULL) {
43:     code.push_back(0);//добавляем в вектор ноль если слева
44:     BuildTable(root->left);//рекурсируем налево по дереву
45:   }
46:
47:   if(root->right!=NULL) {
48:     code.push_back(1);//добавлем в контейнер 1 если справа
49:     BuildTable(root->right);//рекурсируем направо по дереву
50:   }
51:   if (root->left == NULL && root->right == NULL) {//если и слева и справа
    ничего нет
```

```
52:     table[root->c]=code;//то это вершина и приписываем данный полный код
    символу в таблице
53: }
54:     code.pop_back();//очищаем вектор
55: }
56:
57: void encode(void) { //функция кодирования файла
58:   ifstream f(name, ios::binary); // открываем поток для чтения файла в
    двоимчном режиме
59:   list<Node*> t; //объявление двусвязного списка для
60:   map<unsigned char,int> m;// а также вспомогательную таблицу
61:   char c;
62:   while (f.get(c))//подсчет частаты встречаемости символа в файле
63:     m[c]++;
64: ////// запись начальных узлов в список
65:   for(map<unsigned char,int>::iterator itr=m.begin(); itr!=m.end(); ++itr) {
    //установка итератор на начало
66:   //и будем проход по таблице до конца
67:     Node*p = new Node;//выделение памяти под узел и установка его ссылок на
    ноль
68:     p->c = itr->first;//запись в объект в поле char c (будущий узел дерева)
    значения символа
69:     p->a = itr->second;// и ключа к нему в поле int a
70:     t.push_back(p); // добавление элемента в список
71:    }
72: //////построение дерева Хаффмана
73:   while (t.size()!=1) { //пока размер контейнера не равен 1
74:     t.sort(MyCompare());//сортируем список по возрастанию
75:     Node *SonL = t.front();//установка ссылки левого сына на начало
76:     t.pop_front();//удаление начального эелемента и ссылки
77:     Node *SonR = t.front(); //аналогично для правого
78:     t.pop_front();
79:     Node *parent = new Node(SonL,SonR); //выделение памяти под узел и
    установка его ссылок на левого и правого сына
80:     t.push_back(parent);//добавление элемента контейнера к родительскому узлу
81:   }
82:   root = t.front();//корень дерева, установка начала дерева на корень

83:   BuildTable(root);//вызов функции построения таблицы
84:   f.clear();//очистка потока файла name
85:   f.seekg(0); //установка указателя на начало файла
86:   strcat(name1,"_compressed.txt");//добавка к имени начального файла строки
    "сжатый" и расширения .txt
87:   ofstream g(name1, ios::binary);//создание файла с таким именем и открытие
    его потока для ввода в двоичном режиме
88:   while(f.get(c)) {//пока не достигнут конец файла name
89:     vector<bool> x = table[c];//запись вектор префиксного кода каждого
    элемента
90:     for(int n=0; n<x.size(); n++) {
91:       buf = buf | x[n]<<(7-count);//присвоение символа префисному коду в
    кодировке Хаффмана
92:       count++;//счетчик
93:       if (count==8) { //при достижении 8 бита
94:         count=0;//обнуление счетчика
```

```
 95:         g.put(buf);//запись в архив код символа
 96:         buf=0;//обнуление "символа-буфера"
 97:       }
 98:     }
 99:   }
100:   f.close();//закрытием потока начального файла
101:   g.close();//закрытие потока на конечный архив
102: }
103:
104: void decode(void) {//функция декодировки
105:   ifstream F(name1, ios::binary); //открытие потока вывода из архива
106:   strcat(name2,"_decompressed.txt");//добавка к имени начального файла строки
     "разжатый" и расширения .txt
107:   ofstream res(name2, ios::binary);//создание файла с таким именем и открытие
     его потока ввода в двоичном режиме
108:   Node *p = root;//присвоение корня объекту p
109:   count=0;//обнуление счетчика
110:   unsigned char byte;//объявление символа-байта
111:   byte = F.get();//инициализация байта значением сивола в архиве
112:
113:   while(!F.eof()) {//пока не достигнут конец файла
114:     bool b = byte & (1 << (7-count) );//инициализация булевской переменной
115:       if(b) p=p->right;//если результат предыдущей операции 1, то узлу
     присваивается значение правого
116:       else p=p->left;//наоборот
117:       if(p->left==NULL && p->right==NULL) {//если достигнут лист,
118:         res.put(p->c);//то в разархивируемый файл идет запись символа
119:         p=root;//установка узла корнем дерева
120:       }
121:       count++;//итерация счетчика
122:       if(count==8) {//при достижении конца бита
123:         count=0;//счетчик обнуляется
124:         byte = F.get();//байт инициализируется значением по текущему указателю
125:       }
126:     }
127:   F.close();//закрытие потока архива
128:   res.close();//закрытие потока разархивируемого файла
129: }
130:
131: int main (int argc, char *argv[])//точка входа в консольной приложение
132: {
133:   setlocale(LC_ALL, "rus");//русская локализация ввода/вывода
134:   cin>>name;//ввод имени файла
135:   strcpy(name1, name); strcpy(name2, name);//копирование имени файла в строки
     name 1 и 2
136:   strcat(name,".txt");//добавка к файлу расширения .txt
137:   encode();//вызов функции кодировки
138:   decode();//вызов функции декодировки
139:   return 0;//возврат нуля в случае успеха
140: }
```