

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**КУРСОВАЯ РАБОТА**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Статическое кодирование и декодирование текстового файла**  
**методами Хаффмана и Фано-Шеннона.**

Студент(ка) гр. 9381

Андрух И.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

## **ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Студентка Андрух И.А.

Группа 9381

Тема работы:

Вариант 2. Статическое кодирование и декодирование текстового файла методами Хаффмана и Фано-Шеннона. Текущий контроль.

Исходные данные:

Файл с текстом для кодирования

Содержание пояснительной записки:

«Содержание»

«Введение»

«Ход выполнения»

«Тестирование»

«Заключение»

«Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 16.12.2020

Дата защиты реферата: 17.12.2020

Студентка

Андрух И.А.

Преподаватель

Фирсов М.А.

## **АННОТАЦИЯ**

Задача курсовой работы состоит в разработке программы для генерации заданий на кодирование и декодирование текста методами Хаффмана и Фано-Шеннона. Задания предназначены для текущего контроля студентов.

Курсовая работа состоит из пояснительной записки и исходного кода программы.

В ходе работы была реализована программа с консольным интерфейсом для генерации заданий на кодирование и декодирование текстовых файлов методами Хаффмана и Фано-Шеннона. Использовался язык C++.

## **SUMMARY**

The task of the course work is to develop a program for generating tasks for encoding and decoding text using Huffman and Fano-Shannon methods. Tasks are intended for the current control of students.

The course work consists of an explanatory note and the source code of the program.

In the course of the work, a program with a console interface was implemented for generating tasks for encoding and decoding text files using huffman and fano-shannon methods. The C++ language was used.

## СОДЕРЖАНИЕ

	Введение	5
1.	Задание	6
2.	Описание алгоритма	6
3.	Описание структур данных и функций.	7
3.1	struct s_expr	7
3.2	struct binTree	7
3.3	Функции пространства имен binTree_modul	7
3.4	struct elem_code	8
3.5	struct list	8
3.6	struct haffm_codes	8
3.6.1	Поле	8
3.6.2	Методы	8
3.7	Функции для кодирования и декодирования.	9
4.	Описание интерфейса пользователя.	11
5.	Заключение	11
6.	Список литературы	12
7.	Приложение А. Тестирование.	13
	Приложение Б. исходный код программы.	43

## **ВВЕДЕНИЕ.**

Целью работы является написание программы, которая будет генерировать задания с ответами. Задания должны содержать текст для кодирования/декодирования. Кодирование должно быть реализовано по алгоритмам Хаффмана и Фано-Шеннона. В результате работы программы должен генерироваться файл с заданиями и ответами, готовый для передачи студентам.

Алгоритм кодирования Хаффмана: на вход приходят упорядоченные по невозрастанию частот данные, выбираются две наименьших по частоте буквы алфавита, создается родитель(сумма частот этих «листочков»). Потомки удаляются, вместо них записывается родитель, «ветви» родителя нумеруются «0» и «1». Процесс повторяется, пока не будет найден главный родитель – корень.

Алгоритм кодирования Фано-Шеннона: на вход приходят упорядоченные по невозрастанию частот данные, находится середина, которая делит алфавит примерно на две части(примерно равные по сумме частот символов), частям присваивается «0» и «1». Процесс повторяется, пока не будет получен листок.

## **1. ЗАДАНИЕ.**

Реализовать алгоритмы статического кодирования и декодирования текстового файла методами Хаффмана и Фано-Шеннона, генерируя с их помощью задания с ответами для текущего контроля студентов.

## **2. ОПИСАНИЕ АЛГОРИТМА.**

При помощи консольного меню программа запрашивает у пользователя тип генерируемого задания и количество вариантов. После этого берётся строка из заданного изначально массива (преподаватель может менять его, дополняя новыми предложениями) и передается сначала в функцию для добавления символа и частоты его встречаемости в вектор, потом в функцию для сортировки вектора символов, и в функцию кодирования, где происходит создание бинарного дерева одним из двух алгоритмов и сам процесс кодирования. Промежуточные данные, полученные в ходе сортировки, построения дерева и кодирования строки, выводятся в консоль. Таким образом генерируется задание на кодирование строки. Далее генерируется задание на декодирование - выводится закодированная строка и таблица символов(она создается при помощи созданного ранее вектора). Функция декодирования получает закодированную строку и демонстрирует процесс декодирования в консоль, ответ также записывается в файл. Выводится консольное меню для генерации новых заданий или выхода из программы.

### 3. ОПИСАНИЕ СТРУКТУР ДАННЫХ И ФУНКЦИЙ.

#### 3.1 struct s\_expr.

Структура для создания бинарного дерева(пространство имен binTree\_modul) .

```
struct s_expr {  
    bool tag; // true если Leaf, false если Inter  
    union //идентификаторы полей попадают во внешнюю область  
    видимости  
    {  
        base leaf; // лист дерева  
        binTree inter; //звено дерева(не лист)  
    } node;  
};
```

#### 3.2 struct binTree.

Структура бинарного дерева(пространство имен binTree\_modul).

```
struct binTree  
{  
    s_expr* lt; // указатель на левое поддерево  
    s_expr* rt; // указатель на правое поддерево  
};
```

#### 3.3 Функции пространства имен binTree\_modul.

```
lisp create(void); // создание пустого бинарного дерева  
lisp left(const lisp); // возвращает указатель на левое поддерево, аргумент –  
элемент типа s_expr  
lisp right(const lisp); // возвращает указатель на правое поддерево, аргумент  
– элемент типа s_expr  
lisp consBT(const lisp lt, const lisp rt); // создание дерева из двух  
поддеревьев, аргументы –два элемента типа s_expr  
void destroy(lisp&); // уничтожает дерево, аргумент –адрес элемента типа  
s_expr
```

`bool isLeaf(const lisp s);` // проверяет, является ли элемент листом, аргумент – элемент типа `s_expr`

`lisp makeLeaf(const base x);` // создает лист со значением `x`, аргумент – переменная шаблонного типа.

### **3.4 struct elem\_code.**

Структура для хранения символа и соответствующего ему кода.

```
struct elem_code {  
    char symbol;//символ  
    string code;//строка для хранения кода символа  
};
```

### **3.5 struct list.**

Структура для хранения символа и количества повторений этого символа в строке.

```
struct list  
{  
    char element; //символ  
    int count;//количество повторений символа в предложении  
    lisp bt;//элемент типа s_expr, хранит информацию о том, является ли  
элемент листом  
};
```

### **3.6. struct haffm\_codes.**

Структура для хранения кодов символов, полученных по Хаффману.

#### **3.6.1. Поле.**

`vector<elem_code> codes;` - вектор для хранения символов и кодов;

#### **3.6.2. Методы.**

`char get_symbol(string code)` - декодирование символа, аргумент – строка , содержащая код символа;

`string get_code(char symbol)` – получение кода символа, аргумент – символ для кодирования;



`void append(char symbol, string code)` – добавление символа и кода в поле `codes`, аргументы – символ и код символа;

`string encodeString(string plain)` – кодирование строки методом Хаффмана(если код для встреченного символа есть, записываем его в строку результата), аргумент – строка для кодирования;

`void decode_H(string str_to_decode, const lisp b, string& decoded_str)` - декодирование строки методом Хаффмана, аргументы – строка для декодирования, бинарное дерево и строка для результата соответственно;

### **3.7. Функции для кодирования и декодирования.**

- `void add(vector<list>& symbols, char s)` - функция добавления в вектор символа и частоты его встречаемости, аргументы – вектор и символ для добавления;
- `void sort(vector<list>& symbols)` - сортировка символов вектора в порядке невозрастания частот, аргумент – вектор с символами и частотами;
- `void middle(vector<list> simbols, int S , int B , int& node, int& SL, int& SR)`- деление символов на две части по сумме частот для алгоритма Фано-Шеннона, аргументы – вектор возможных символов, длина сообщения, начало поиска, счетчик добавленных в левую часть символов, сумма частот левой части, сумма частот правой части;
- `void method_FSH(vector<list> smbl, int lenght, int start_of_search, int end_of_search, string Code, lisp& bin_t,int deep)` – создание бинарного дерева для метода Фано-Шеннона, аргументы- вектор символов, длина сообщения, начало обрабатываемого интервала, конец обрабатываемого интервала, строка для формирования кода, бинарное дерево, глубина рекурсии;
- `void show_method_FSH(vector<list> smbl, int lenght, int start_of_search, int end_of_search, string Code, lisp& bin_t,int deep)` – функция, аналогичная предыдущей, но без демонстрации промежуточных результатов;

- `void final_coding(vector<list>& smbl, char elm, string& coded_string, int& l, string codes[])` - функция кодирования элемента и записи его в результирующую строку, для метода Фано-Шеннона, аргументы – вектор символов, элемент для кодирования, строка результата, длина строки результата, массив строк, хранящих коды символов;
- `void close_final_coding(vector<list>& smbl, char elm, string& coded_string, int& l, string codes[])` – функция, аналогичная предыдущей, но без вывода промежуточных результатов;
- `void method_H(vector<list>& smbl, list& bin_tree, string& code)` - создание бинарного дерева элементов по Хаффману, аргументы – вектор символов, бинарное дерево, код символа;
- `void search_of_min_element(vector<list>& smbl, list& elmn)` - поиск элемента с минимальной частотой повторений, аргументы – вектор символов, проверяемый элемент;
- `void codeHuffman(string& code, vector<list>& smbl, list bin_tree, int deep)` - создание бинарного дерева кодов по Хаффману, аргументы – строка с уже полученным кодом, вектор символов, бинарное дерево, глубина рекурсии;
- `void decode_FSH(vector<list> smbl, string str_to_decode, const list b, string& decoded_str)` - функция декодирования, аргументы – вектор символов, строка для декодирования, бинарное дерево, строка для результата.

#### **4. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ.**

Для взаимодействия пользователя и программы выводится консольное меню. Пользователь может выбрать тип задания(Кодирование по Хаффману и декодирование, или кодирование по Фано-Шеннону и декодирование). Далее предлагается выбрать количество генерируемых вариантов. Задания и ответы выводятся в файл. Задания, промежуточные результаты и ответы для проверки выводятся в консоль.

#### **ЗАКЛЮЧЕНИЕ.**

В результате выполнения курсовой работы была написана программа на языке C++ для генерации заданий на статическое кодирование методами Хаффмана и Фано-Шеннона и декодирование. Задания и ответы сохраняются в файл и готовы для передачи студентом. Массив возможных строк для обработки может обновляться и дополняться преподавателем.

## **СПИСОК ЛИТЕРАТУРЫ.**

1. Керниган Б. И Ритчи Д. Язык программирования Си.
2. <https://habr.com/ru/post/438512/>
3. <https://habr.com/ru/post/137766/>

# **ПРИЛОЖЕНИЕ А** **ТЕСТИРОВАНИЕ.**

№	Вх од ны е дан ны е	Результат
1	1	ВАРИАНТ №1
	1	<p>Задание 1.</p> <p>Закодируйте предложение методом Хаффмана:&lt;&lt;</p> <p>peter piper picked a peck of pickled peppers</p> <p>Промежуточные выводы:</p> <p>Символы отсортированы по частоте встречаемости:</p> <p>p-9</p> <p>e-8</p> <p>-7</p> <p>r-3</p> <p>i-3</p> <p>c-3</p> <p>k-3</p> <p>d-2</p> <p>t-1</p> <p>a-1</p> <p>o-1</p> <p>f-1</p> <p>l-1</p> <p>s-1</p>

Построение бинарного дерева:

0

00

000

0000

0001

00010

00011

001

0010

00100

00101

0011

00110

00111

01

1

10

100

1000

1001

101

1010

1011

11

110

111

Процесс кодирования:

		p-01
		e-111
		t-00010
		e-111
		r-1000
		-110
		p-01
		i-1001
		p-01
		e-111
		r-1000
		-110
		p-01
		i-1001
		c-1010
		k-1011
		e-111
		d-0000
		-110
		a-00011
		-110
		p-01
		e-111
		c-1010
		k-1011
		-110
		o-00100
		f-00101

-110

p-01

i-1001

c-1010

k-1011

l-00110

e-111

d-0000

-110

p-01

e-111

p-01

p-01

e-111

r-1000

s-00111

Ответ:

01111000101111000110011001011111000110011001101010111110000  
11000011110011111010101111000100001011100110011010101100110  
1110000110011110101111100000111

Задание 2.

Раскодируйте строку:

00010111000101000110010100001010001100101010111100010110101  
1111010011000101011110001111101111100110010101011110011111  
00101101011000100000010100111111

Коды символов:

-----



	<p>p   00</p> <p>e   010</p> <p>    011</p> <p>r   100</p> <p>i   1010</p> <p>c   1011</p> <p>k   1100</p> <p>d   1101</p> <p>t   11100</p> <p>a   111010</p> <p>o   111011</p> <p>f   11110</p> <p>l   111110</p> <p>s   111111</p> <p>Процесс декодирования:</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>е</p> <p>Встречен '1'</p>
--	--

	<p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>t</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>e</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>г</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p>
--	---

	<p>р</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>і</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>е</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>г</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p>
--	--

	<p>Декодирован символ:</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>і</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>с</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>к</p> <p>Встречен '0'</p>
--	--

	<p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>е</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>d</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>а</p> <p>Встречен '0'</p> <p>Встречен '1'</p>
--	--

	<p>Встречен '1'</p> <p>Декодирован символ:</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>е</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>с</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>к</p>
--	--

	<p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>o</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>f</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p>
--	---

	<p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>і</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>с</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>к</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p>
--	---



	<p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>1</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>е</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>d</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p>
--	---

	<p>р</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>е</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>е</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>г</p>
--	---

		<p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>s</p> <p>Ответ:</p> <p>peter piper picked a peck of pickled peppers</p> <p>Варианты записаны в файл fout.txt</p>
2	2	ВАРИАНТ №1
	1	<p>Задание 1.</p> <p>Закодируйте предложение методом Фано - Шеннона:</p> <p>peter piper picked a peck of pickled peppers</p> <p>Символы отсортированы по частоте встречаемости:</p> <p>p-9</p> <p>e-8</p> <p>-7</p> <p>r-3</p> <p>i-3</p> <p>c-3</p> <p>k-3</p> <p>d-2</p> <p>t-1</p> <p>a-1</p> <p>o-1</p>

	f-1
	l-1
	s-1
	Построение бинарного дерева Шеннона-Фано:
	0
	00
	01
	010
	011
	1
	10
	100
	101
	1010
	1011
	11
	110
	1100
	1101
	111
	1110
	11100
	11101
	111010
	111011
	1111
	11110
	11111

		111110 111111  Коды символов: ----- p 00 e 010  011 r 100 i 1010 c 1011 k 1100 d 1101 t 11100 a 111010 o 111011 f 11110 l 111110 s 111111  Процесс кодирования: символ р заменён на код 00 символ е заменён на код 010 символ t заменён на код 11100 символ e заменён на код 010 символ r заменён на код 100 символ  заменён на код 011 символ р заменён на код 00 символ i заменён на код 1010
--	--	---

	символ p заменён на код 00
	символ e заменён на код 010
	символ r заменён на код 100
	символ    заменён на код 011
	символ p заменён на код 00
	символ i заменён на код 1010
	символ c заменён на код 1011
	символ k заменён на код 1100
	символ e заменён на код 010
	символ d заменён на код 1101
	символ    заменён на код 011
	символ a заменён на код 111010
	символ    заменён на код 011
	символ p заменён на код 00
	символ e заменён на код 010
	символ c заменён на код 1011
	символ k заменён на код 1100
	символ    заменён на код 011
	символ o заменён на код 111011
	символ f заменён на код 11110
	символ    заменён на код 011
	символ p заменён на код 00
	символ i заменён на код 1010
	символ c заменён на код 1011
	символ k заменён на код 1100
	символ l заменён на код 111110
	символ e заменён на код 010
	символ d заменён на код 1101
	символ    заменён на код 011

	<p>символ р заменён на код 00</p> <p>символ е заменён на код 010</p> <p>символ р заменён на код 00</p> <p>символ р заменён на код 00</p> <p>символ е заменён на код 010</p> <p>символ г заменён на код 100</p> <p>символ s заменён на код 11111</p> <p>Ответ:</p> <p>00010111000101000110010100001010001100101010111100010110101  1111010011000101011110001111011111100110010101011110011111  00101101011000100000010100111111</p> <p>Задание 2.</p> <p>Раскодируйте строку:</p> <p>00010111000101000110010100001010001100101010111100010110101  1111010011000101011110001111011111100110010101011110011111  00101101011000100000010100111111</p> <p>Коды символов:</p> <p>-----</p> <p>p 00</p> <p>e 010</p> <p> 011</p> <p>r 100</p> <p>i 1010</p> <p>c 1011</p> <p>k 1100</p> <p>d 1101</p>
--	--

	<p>t 11100</p> <p>a 111010</p> <p>o 111011</p> <p>f 11110</p> <p>l 111110</p> <p>s 111111</p> <p>Процесс декодирования:</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>е</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>t</p> <p>Встречен '0'</p>
--	---



	<p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>е</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>г</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>і</p>
--	--

	<p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>е</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>г</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '1'</p>
--	--

	<p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>i</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>c</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>k</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>e</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p>
--	---

	<p>Встречен '1'</p> <p>Декодирован символ:</p> <p>d</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>a</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p>
--	--

	<p>р</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>е</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>с</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>к</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>Встречен '1'</p> <p>Встречен '1'</p>
--	--

	<p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>о</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>f</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p>
--	--

	<p>Встречен '0'</p> <p>Декодирован символ:</p> <p>i</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>c</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>k</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>l</p> <p>Встречен '0'</p> <p>Встречен '1'</p>
--	---

	<p>Встречен '0'</p> <p>Декодирован символ:</p> <p>е</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>d</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>е</p> <p>Встречен '0'</p>
--	--



	<p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>р</p> <p>Встречен '0'</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>е</p> <p>Встречен '1'</p> <p>Встречен '0'</p> <p>Встречен '0'</p> <p>Декодирован символ:</p> <p>г</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Встречен '1'</p> <p>Декодирован символ:</p>
--	--

		<p>s</p> <p>Ответ: peter piper picked a peck of pickled peppers</p> <p>Варианты записаны в файл fout.txt</p>
--	--	--

## ПРИЛОЖЕНИЕ Б

### ИСХОДНЫЙ КОД ПРОГРАММЫ

#### Файл main.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <stdlib.h>
#include <locale.h>
#include "header.h"
using namespace std;
using namespace binTree_modul;

struct elem_code { //структура для хранения символа и соответствующего ему
кода
    char symbol;
    string code;
};

struct haffm_codes {
    vector<elem_code> codes; //вектор для хранения кодов символов при
кодировании по Хаффману
    char get_symbol(string code) { //декодирование строки
        for (vector<elem_code>::iterator it = codes.begin(); it !=
codes.end(); it++) {
            if ((*it).code == code) {
                return (*it).symbol;
            }
        }
        return '\\0';
    };
    string get_code(char symbol) { //поиск кода для символа
        for (vector<elem_code>::iterator it = codes.begin(); it !=
codes.end(); it++) {
            if ((*it).symbol == symbol) {
```

```

        return (*it).code;
    }
}
return "";
};

void append(char symbol, string code) { //добавление символа и кода в
codes
    elem_code new_code;
    new_code.code = code;
    new_code.symbol = symbol;
    codes.push_back(new_code);
};

string encodeString(string plain) { //кодирование строки методом Хаффмана
    string result = "";
    string step_code = "";
    cout<<"Процесс кодирования:"<<endl;
    for (int i = 0; i < plain.length(); i++) {
        step_code = get_code(plain[i]);
        if (step_code != "") { //Если код для встреченного
символа есть, записываем его в строку результата
            result +=step_code;
            cout << "\n" <<plain[i]<<"-" <<  step_code;
        }
        else {

        }
    }
    return result;
};

void decode_H(string str_to_decode, const lisp b, string&
decoded_str) //декодирование строки методом Хаффмана
{

    string curr_code = "";
    decoded_str = "";
    int i = 0;

```

```

while (i <= str_to_decode.size() - 1)
{
    lisp p = b;
    curr_code = "";

    while (!isLeaf(p))
    {
        if (i >= str_to_decode.size())
            break;

        curr_code += str_to_decode[i];
        i++;

        if (curr_code[curr_code.length() - 1] == '0') p = Left(p);
//если встречен 0 спускаемся по левой ветке дерева
        else p = Right(p); //иначе - по правой
    }

    decoded_str += get_symbol(curr_code); //дойдя до листа извлекаем
из него символ
}

};

};

struct list
{
    char element; //символ
    int count; //количество повторений символа в предложении
    lisp bt;
};

haffm_codes h_codes;

string code_of_elementFSH[50]; //массив для хранения кодов символов по Шеннону

void add(vector<list>& symbols, char s);
void sort(vector<list>& symbols);

```

```

void method_FSH(vector<list> smbl, int lenght, int start_of_search, int
end_of_search, string Code, lisp& bin_t,int deep);

void show_method_FSH(vector<list> smbl, int lenght, int start_of_search, int
end_of_search, string Code, lisp& bin_t,int deep);

void final_coding(vector<list>& smbl, char elm, string& coded_string, int& l,
string codes[]);

void close_final_coding(vector<list>& smbl, char elm, string& coded_string,
int& l, string codes[]);

void decode_FSH(vector<list> smbl, string str_to_decode, const lisp b,
string& decoded_str);

void method_H(vector<list>& smbl, lisp& bin_tree, string& code);

void search_of_min_element(vector<list>& smbl, list& elmn);

void codeHuffman(string& code, vector<list>& smbl, lisp bin_tree,int deep);

const char* ar[] = { "peter piper picked a peck of pickled peppers", "the
shells she sells are the sea-shore shells", "better than the bitter butter",
                        "i thought a thought", "sea-serpents swam the seven
seas", "she sits she shines"};

```

```

int main()
{
    setlocale(LC_ALL,"rus");
    ofstream fout("/home/ira/CLionProjects/AiSD_course_work/fout.txt");
    unsigned int count_variant;
    int action;
    bool exit = false;
    while (!exit)
    {
        cout << "Выберите тип варианта: " << endl;
        cout << "1.  Кодирование Хаффмана и декодирование Фано - Шеннона." <<
endl;
        cout << "2.  Кодирование Фано - Шеннона и декодирование Хаффмана. "
<< endl;
        cout << "3.  Выход. \n -----" << endl;
        cin >> action;
        switch (action)
        {
            case 1:

```

```

cout << "Здравствуйте! Введите количество нужных вариантов: "
<< endl;

cin >> count_variant;
for (int i = 0; i < count_variant; i++)
{

    fout << " ВАРИАНТ №" << i + 1 << endl
        << "Задание 1.\nЗакодируйте предложение методом
Хаффмана:  << ";

    cout << " ВАРИАНТ №" << i + 1 << endl
        << "Задание 1.\nЗакодируйте предложение методом
Хаффмана:  << "<<endl;

    string main_str = ar[i]; //берем строку из массива
    cout<<main_str;
    fout<<main_str;
    int lenght_of_main_str = main_str.size();
    cout<<"\nПромежуточные выводы:\n";
    vector<list> symbolsH;//вектор для метода Хаффмана
    vector<list> symbolsFSH;//вектор для частот и
символов, дерева Фано-Шеннон
    lisp binary_tree = Create();//дерево для кодирования и
декодирования

    lisp binary_treeH = Create();

    string codeFSH;//код символа,будет помещен в массив
code_of_element..

    string codeH;
    string coded_stringFSH;//закодированная строка
    string coded_stringH;
    string decodedFSH;//декодированная строка
    string decodeH;
    string curr_str;//дополнительная строка для заполнения
вектора

    curr_str = string(main_str);

    for (int i = 0; i < curr_str.length(); i++) //добавление
в вектор символа и частоты встречаемости этого символа
    {
        add(symbolsFSH, curr_str[i]);
        add(symbolsH, curr_str[i]);
    }
}

```

```

        sort(symbolsFSH); //сортируем символы для Шеннона-Фано в
порядке невозрастания частот
        cout<<"Символы отсортированы по частоте
встречаемости:\n";
        int k = 0;
        sort(symbolsH); //сортируем символы для Хаффмана в порядке
невозрастания частот
        while(symbolsH[k].element) {
            cout << symbolsH[k].element << "-" <<
symbolsH[k].count << endl;
            k++;
        }
        //далее создаем бинарное дерево кодов по алгоритму
Шеннона-Фано
        method_FSH(symbolsFSH, lenght_of_main_str, 0,
symbolsFSH.size() - 1, codeFSH, binary_tree,0);
        for (int i = 0, l = 0; i < lenght_of_main_str; i++)
        {
            close_final_coding(symbolsFSH, main_str[i],
coded_stringFSH, l, code_of_elementFSH); // Кодирование строки Ф-Ш
        }
        cout<<"\nПостроение бинарного дерева: "<<endl;
        method_H(symbolsH, binary_treeH, codeH); //создаем
бин.дерево символов по Хаффману
        codeHuffman(codeH, symbolsH, binary_treeH,0); //по дереву
получаем коды символов
        coded_stringH = h_codes.encodeString(main_str); //кодируем
строку методом Хаффмана
        cout<<"\nОтвет:\n";
        h_codes.decode_H(coded_stringH, binary_treeH, decodeH);
        cout << coded_stringH << endl
            << "\nЗадание 2.\nРаскодируйте строку:" << endl;
        cout << coded_stringFSH << endl << endl;
        cout << "Коды символов: " << endl;
        cout << "-----" << endl;
        for (int i = 0; i < symbolsFSH.size(); i++)
        {
            cout << " " << symbolsFSH[i].element << " | " <<
code_of_elementFSH[i] << endl;
        }
        string str;

```



```

        fout << main_str << ">>" << endl
        << "\nЗадание 2.\nРаскодируйте строку:" << endl;
        fout << coded_stringFSH << endl << endl;
        fout << "\nКоды символов: " << endl;
        fout << "-----" << endl;
        for (int i = 0; i < symbolsFSH.size(); i++)
        {
            fout << " " << symbolsFSH[i].element << " | " <<
code_of_elementFSH[i] << endl;
        }
        cout << "\nПроцесс декодирования:\n"<<endl;

        decode_FSH(symbolsFSH,coded_stringFSH,binary_tree,str);//декодируем строку
        cout<<"\nОтвет:\n"<<main_str<<endl;
        fout << "-----" << endl
        -----" << endl
            << "\n      Ответы.      Вариант №" << i + 1 << endl
            << "\nЗадание 1.\n" << "Метод Хаффмана:\n" <<
coded_stringH << endl;
            fout << "\nЗадание 2.\n" << endl << "<<" << main_str <<
">>" << endl;//раскодированная строка

            fout <<
"
-----
-----\n\n" << endl;
        }
        cout << "Варианты записаны в файл fout.txt" << endl;

        break;
    case 2:

        cout << "Здравствуйте! Введите количество нужных вариантов: "
<< endl;

        cin >> count_variant;
        for (int i = 0; i < count_variant; i++)
        {
            string main_str = ar[i];
            fout << " ВАРИАНТ №" << i + 1 << endl
                << "Задание 1.\nЗакодируйте предложение методом Фано
- Шеннона: \n "<<main_str<<endl;

```

```

        cout << "ВАРИАНТ №" << i + 1 << endl

        << "Задание 1.\nЗакодируйте предложение методом Фано
- Шеннона:  \n "<<main_str<<endl;

        int lenght_of_main_str = main_str.size();

        vector<list> symbolsH;//вектор для метода Хаффмана
        vector<list> symbolsFSH;//вектор для частот и
символов, дерева Фано-Шеннон
        lisp binary_tree = Create();//дерево для кодирования и
декодирования
        lisp binary_treeH = Create();
        string codeFSH;//код символа,будет помещен в массив
code_of_element..
        string codeH;
        string coded_stringFSH;//закодированная строка
        string coded_stringH;
        string decodedFSH;
        string decodeH;
        string curr_str;//дополнительная строка для заполнения
вектора
        curr_str = string(main_str);
        for (int i = 0; i < curr_str.length(); i++) //добавление
в вектор символа и частоты
        {
            add(symbolsFSH, curr_str[i]);
            add(symbolsH, curr_str[i]);

        }

        cout<<"Символы отсортированы по частоте
встречаемости:"<<endl;

        sort(symbolsFSH);//сортируем символы в порядке
невозрастания частот
        int k;
        while(symbolsFSH[k].element){
            cout << symbolsFSH[k].element << "-" <<
symbolsFSH[k].count << endl;
            k++;
        }

        cout<<"\nПостроение бинарного дерева Шеннона-Фано:
"<<endl;

        //строим бинарное дерево по Шеннону-Фано

```

```

        show_method_FSH(symbolsFSH, lenght_of_main_str, 0,
symbolsFSH.size() - 1, codeFSH, binary_tree,0);

        cout << "\nКоды символов: " << endl;
        cout << "-----" << endl;
        for (int i = 0; i < symbolsFSH.size(); i++)
        {
            cout << " " << symbolsFSH[i].element << "|" <<
code_of_elementFSH[i] << endl;
        }

        cout<<"\nПроцесс кодирования:"<<endl;
        for (int i = 0, l = 0; i < lenght_of_main_str; i++)
        {
            final_coding(symbolsFSH, main_str[i],
coded_stringFSH, l, code_of_elementFSH); // Кодирование строки Ф-Ш
        }

        cout<<"\nОтвет: "<<coded_stringFSH<<endl;
        cout << "\nЗадание 2.\nРаскодируйте строку:" << endl;
        cout << coded_stringFSH << endl << endl;
        cout << "Коды символов: " << endl;
        cout << "-----" << endl;
        for (int i = 0; i < symbolsFSH.size(); i++)
        {
            cout << " " << symbolsFSH[i].element << "|" <<
code_of_elementFSH[i] << endl;
        }

        fout << "\nЗадание 2.\nРаскодируйте строку:" << endl;
        fout << coded_stringFSH << endl << endl;
        fout << "Коды символов: " << endl;
        fout << "-----" << endl;

        string str;
        for (int i = 0; i < symbolsFSH.size(); i++)
        {
            fout << " " << symbolsFSH[i].element << "|" <<
code_of_elementFSH[i] << endl;
        }

        cout << "\nПроцесс декодирования:\n"<<endl;

        decode_FSH(symbolsFSH,coded_stringFSH,binary_tree,str);//декодирование строки

        cout<<"Ответ: "<<main_str<<endl;

```

```

        fout << "-----"
-----" << endl

        << "\n    Ответы.    Вариант №" << i + 1 << endl
        << "\nЗадание 1.\n" << "Метод Фано-Шеннон:\n" <<
coded_stringFSH << endl// << "Метод Хаффмана:\n" << coded_stringH << endl
        << "\nЗадание 2.\n" << endl << "<<" << main_str <<
">>" << endl;//раскодированная строка

        fout <<
"
-----\n\n" << endl;
    }

    cout << "Варианты записаны в файл fout.txt \n -----"
-" << endl;

    break;

    case 3:
        exit = true;
        break;
    default:
        cout << "Incorect num \n -----" << endl;
        break;
    }

}

system("pause");
}

```

```

void add(vector<list>& symbols, char s)//функция добавления в вектор символа
и частоты его встречаемости

```

```

{
    bool isFound = false;
    for (int i = 0; i < symbols.size(); i++) {
        if (symbols[i].element == s) {
            symbols[i].count++;
            isFound = true;
            break;
        }
    }
}

```

```

        }
    }
    if (!isFound) {
        list e;
        e.element = s;
        e.count = 1;
        symbols.push_back(e);
    }
}

```

void sort(vector<list>& symbols)//сортировка символов вектора в порядке невозрастания частот

```

{
    int size = symbols.size();
    for (int i = 0; i < size - 1; i++)
    {
        for (int j = 0; j < size - i - 1; j++)
        {
            if (symbols[j + 1].count > symbols[j].count)
            {
                swap(symbols[j], symbols[j + 1]);
            }
        }
    }
}

```

void middle(vector<list> symbols, int S , int B , int& node, int& SL, int& SR)//деление символов на две части по сумме частот

```

{

    int D;
    node = B;
    SL = symbols[node].count;
    SR = S - SL;
    D = SL - SR;
    while ((D + symbols[node + 1].count) < 0) {
        node++;
    }
}

```

```

        SL = SL + symbols[node].count;
        SR = SR - symbols[node].count;
        D = SL - SR;
    }
}

//демонстрация создания дерева по Шеннону-Фано
void show_method_FSH(vector<list> symb1, int lenght, int start_of_search, int
end_of_search, string Code, list& bin_t,int deep)
{
    int node;
    int sum_of_Lt, sum_of_Rt;
    if (start_of_search == end_of_search)//если 1 элемент
    {
        bin_t = makeLeaf(start_of_search);
        code_of_elementFSH[start_of_search] = Code;
    }
    else {
        list bL = Create();
        list bR = Create();
        middle(symb1, lenght, start_of_search, node, sum_of_Lt,
sum_of_Rt);//находим середину по сумме частот
        string str1 = Code;
        string str2 = Code;
        str1.append("0");//в код левой части добавляем 0
        str2.append("1");//в код правой части добавляем 1
        for(int i=0;i<deep;i++)
            cout<<" ";
        cout<<str1<<endl;
        show_method_FSH(symb1, sum_of_Lt, start_of_search, node, str1,
bL,deep+1);//рекурсивный вызов для каждой из частей
        for(int i=0;i<deep;i++)
            cout<<" ";
        cout<<str2<<endl;
        show_method_FSH(symb1, sum_of_Rt, node + 1, end_of_search, str2,
bR,deep+1);
        bin_t = ConsBT(bL, bR);
    }
}

```

```

//создание дерева по методу Шеннона-Фано без демонстрации
void method_FSH(vector<list> smbl, int lenght, int start_of_search, int
end_of_search, string Code, lisp& bin_t,int deep)
{
    int node;
    int sum_of_Lt, sum_of_Rt;
    if (start_of_search == end_of_search)//если 1 элемент
    {
        bin_t = makeLeaf(start_of_search);
        code_of_elementFSH[start_of_search] = Code;
    }
    else {
        lisp bL = Create();
        lisp bR = Create();
        middle(smbl, lenght, start_of_search, node, sum_of_Lt,
sum_of_Rt);//находим середину по сумме частот
        string str1 = Code;
        string str2 = Code;
        str1.append("0");//в код левой части добавляем 0
        str2.append("1");//в код правой части добавляем 1
        method_FSH(smbl, sum_of_Lt, start_of_search, node, str1,
bL,deep+1);//рекурсивный вызов для каждой из частей
        method_FSH(smbl, sum_of_Rt, node + 1, end_of_search, str2,
bR,deep+1);
        bin_t = ConsBT(bL, bR);
    }
}

void final_coding(vector<list>& smbl, char elm, string& coded_string, int& l,
string codes[])//функция кодирования с демонстрацией
{
    for (int i = 0; i < smbl.size(); i++) {
        if (elm == smbl[i].element)//если встреченный символ равен символу из
вектора
        {
            l += code_of_elementFSH[i].size();

```

```

        coded_string.append(codes[i]); //в закодированную строку добавляем
код этого символа

        cout << "символ " << elm << " заменён на код " << codes[i] <<
endl;

    }

}

}

```

//функция кодирования без демонстрации

```

void close_final_coding(vector<list>& smbl, char elm, string& coded_string,
int& l, string codes[])
{

```

```

    for (int i = 0; i < smbl.size(); i++) {
        if (elm == smbl[i].element)
        {
            l += code_of_elementFSH[i].size();
            coded_string.append(codes[i]);

        }
    }
}

```

void method\_H(vector<list>& smbl, lisp& bin\_tree, string& code) //создаем  
дерево элементов по Хаффману

```

{
    for (int i = 0; i < smbl.size(); i++) { smbl[i].bt =
makeLeaf(smbl[i].element); } //дополним вектор деревом

    while (smbl.size() > 1)
    {
        list min_tree1, min_tree2;
        search_of_min_element(smbl, min_tree1);
        search_of_min_element(smbl, min_tree2); //находим два минимальных
элемента

        list b;

```



```

        b.bt = ConsBT(min_tree1.bt, min_tree2.bt); //создаем поддереву из
минимальных элементов

        b.count = min_tree1.count + min_tree2.count; //складываем частоты

        smbl.push_back(b); //добавляем в вектор получившееся дерево

    }

    bin_tree = smbl[0].bt;

}

void search_of_min_element(vector<list>& smbl, list& elmn) //поиск элемента с
минимальной частотой повторений

{
    int deleteItem = 0, i = 0;
    elmn = smbl[0];
    while (i < smbl.size())
    {
        if (smb1[i].count < elmn.count)
        {
            elmn = smbl[i];
            deleteItem = i;
        }
        i++;
    }

    smbl.erase(smb1.begin() + deleteItem);
}

void codeHuffman(string& code, vector<list>& smbl, list bin_tree, int
deerp) //создаем бинарное дерево кодов по Хаффману

{
    if (isLeaf(bin_tree)) //если дошли до листа, то в h.codes описываем символ
и код
    {
        h_codes.append(bin_tree->node.leaf, code);
    }
    else

```

```

{
    string str1 = code;
    string str2 = code;
    str1.append("0");//для левого поддеревя в код добавляем 0
    str2.append("1");//для правого поддеревя в код добавляем 1
    for(int i=0;i<deep;i++)
        cout<<" ";
    cout<<str1<<endl;
    codeHuffman(str1, smbl, Left(bin_tree),deep+1);//вызываем рекурсивно
    для левого и правого поддеревьев
    for(int i=0;i<deep;i++)
        cout<<" ";
    cout<<str2<<endl;
    codeHuffman(str2, smbl, Right(bin_tree),deep+1);
}
}

```

void decode\_FSH(vector<list> smbl, string str\_to\_decode, const lisp b, string& decoded\_str)//функция декодирования

```

{
    char x;
    int i = 0;
    while (i != str_to_decode.size())
    {
        lisp p = b;
        if (!isLeaf(p))
        {
            while (!isLeaf(p))//пока не дошли до листа
            {
                x = str_to_decode[i];
                i++;
                if (x == '0') { //если встречен 0, спускаемся по левому
поддереву
                    p = Left(p);
                    cout<<"Встречен '0'"<<endl;
                }
                else { //если встречен 1, спускаемся по правому поддереву
                    p = Right(p);

```

```

        cout<<"Встречен '1'"<<endl;
    }
}

    cout <<"\nДекодирован символ:\n"<<smb1[p->node.leaf].element<<endl;
}

    else i++;

    decoded_str.push_back(smb1[p->node.leaf].element); //добавляем элемент
в декодированную строку
}
}

```

## Файл header.h

```

namespace binTree_modul
{

```

```

//-----
-----

typedef int base;    // базовый тип элементов (атомов)

struct s_expr;
struct binTree
{
    s_expr* lt; // указатель на левое поддерево
    s_expr* rt; // указатель на правое поддерево
};

struct s_expr {
    bool tag; // true если Leaf, false если Inter
    union //идентификаторы полей попадают во внешнюю область видимости
    {
        base leaf; // лист дерева
        binTree inter; //звено дерева(не лист)
    } node;
};

typedef s_expr* lisp;

```

```

//-----

lisp Create(void); // создание пустого бинарного дерева
lisp Left(const lisp); // возвращает указатель на левое поддерево
lisp Right(const lisp); // возвращает указатель на правое поддерево
lisp ConsBT(const lisp lt, const lisp rt); // создание дерева из двух
поддеревьев
void destroy(lisp&); // уничтожает дерево
bool isLeaf(const lisp s); // проверяет, является ли элемент листом
lisp makeLeaf(const base x); // создает лист со значением x

}

```

## Файл header.cpp

```

#include <iostream>
#include <cstdlib>
#include "header.h"
using namespace std;

namespace binTree_modul
{
    bool isLeaf(const lisp s)
    {
        if (s == NULL) return false;
        else return (s->tag);
    }

    //-----

    lisp Create()
    {
        return NULL;
    }

    //-----

    lisp Left(const lisp b)
    {
        if (b != NULL) {

```

```

        if (!isLeaf(b)) return b->node.inter.lt;
        else {
            cout << b->node.leaf << "\n";
            return NULL;
        }
    }
}

//-----

lisp Right(lisp b)
{
    if (b != NULL) {
        if (!isLeaf(b)) return b->node.inter.rt;
        else {
            cout << b->node.leaf << "\n";
            return NULL;
        }
    }
}

//-----

lisp ConsBT(const lisp lt, const lisp rt)
{
    lisp p;
    p = new s_expr;
    if (p == NULL) { cerr << "Memory not enough\n"; exit(1); }
    else {
        p->tag = false;
        p->node.inter.lt = lt;
        p->node.inter.rt = rt;
        return p;
    }
}

//-----

void destroy(lisp& b)
{
    if (b != NULL) {
        destroy(b->node.inter.lt);
        destroy(b->node.inter.rt);
    }
}

```

```

        delete b;
        b = NULL;
    }
}

//-----
lisp makeLeaf(const base x)
{
    lisp s;
    s = new s_expr;
    s->tag = true;
    s->node.leaf = x;
    return s;
}

//-----
}

```