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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Алгоритмы кодирования

Студент гр. 9302 _____ Точилин А.Е.

Преподаватель _____ Тутуева А.В.

Санкт-Петербург

2021

Цель работы

Реализовать алгоритм кодирования Шеннона-Фано в C++.

1 Ход работы

1.1 Постановка задачи

Реализовать на основе структур из 1 лабораторной работы, алгоритм кодирования Шеннона-Фано.

Список методов:

void input(); ввод строки
void FindSymbols();//поиск различных элементов строки
void ListSort();// сортировка элементов по количеству повторений в строке
void coding();//кодирование строки
void decoding();//декодирование строки
void print(); //вывод результата работы программы в консоль

1.2 Описание пользовательских типов данных

Использовал класс данных ShenonList, состоящий из 4 элементов типа List<T>: List<char> word, List<char> symbols, List<char> decodedWord, List<bool> encodedWord, представляющие собой односвязные списки, дополненные полем int count и вложенным указателем на List<bool>* code, для создания кодов данных по алгоритму Шеннона-Фано.

1.3 Оценка временной сложности методов

метод	Временная сложность
void input()	$O(n^2)$*
void FindSymbols()	O(n)
void ListSort()	O(nlogn)
void coding()	$O(n^2)$
void decoding()	$O(n^2)$
void print()	O(n)

*так как в ввод входят оставшиеся функции без вывода

1.4 Описание реализованных unit-тестов

Проверил правильность кодирования для слова ‘zx’ получил закодированное слово ‘1 0’, и для слова ‘xxzx5555’ получил закодированное слово ‘10 10 11 10 0 0 0 0’. После этого декодировал оба слова и получил исходные слова. Также проверил нахождения всех символов в этих словах, получил что и ожидал: для первого слова x, z; для второго 5, x, z. Еще проверил

кодирование для длинной последовательности из одного символа и для длинной последовательности 2-ух чередующихся букв, как и ожидалось построение закодированных слов выполнилось успешно.

1.5 Пример работы программы

```
it is test string
Entered word - it is test string
Table of codes
|t - 000
|s - 001
|i - 01
|  - 10
|r - 1100
|n - 1101
|g - 1110
|e - 1111

Encoded word - 01000100100110000111100100010001000110001110111110
Decoded word - it is test string
compression ratio is equal 2.77551
```

1.6 Код программы

List.h

```
#include<iostream>

using namespace std;
template<class T>
class List
{
private:
	class Node {
	public:
		Node(T data = T(), Node* Next = NULL) {
			this->data = data;
			this->Next = Next;
			this->code = NULL;
			this->count = 0;
		}
		Node(T data, int count) :data(data), Next(NULL), code(new List<bool>()),
count(count) {};
		Node* Next;
		T data;
		unsigned short int count;
		List<bool>* code;
	};

	Node* head;
	Node* tail;
	Node* cur;
	int Size;
```

```
public:



	void iterator(Node* curent);//create iterator
	bool HasNext();//we check have next element
	void push_with_count(T obj, int count);//adding in the end with count
	void push_back(T obj); // add elem to back
	void push_front(T obj); // add elem to front
	void pop_back(); // del elem from back
	void pop_front(); // del elem from head
	void insert(T obj, size_t k); // add elem by index, insert before elem with this
index
	T at(size_t k); // get elem by index
	int atCount(size_t k); // geting count of element
	void swap(size_t index_one, size_t index_two);//swap of 2 elements
	void remove(int k); // delete elem by index
	size_t get_size(); // get size
	void print_to_console(); // output list
	void print_to_console_with_code();
	void clear();  // clear list
	void set(size_t k, T obj);  // замена элемента по индексу на передаваемый элемент
//change elem at index
	void set_with_count(size_t k, T obj, int count);
	bool isEmpty(); // has list any elements
	void reverse();  // reverse list
	List(Node* head = NULL, Node* tail = NULL, int Size = 0) :head(head), tail(tail),
Size(Size) {}
	~List() {
		if (head != NULL) {
			this->clear();
		}
	};
	Node* getHead()//geting head
	{
		return this->head;
	}

	Node* Next()//geting current element from iterator, move to next element
	{
		Node* temp;
		temp = cur;
		cur = cur->Next;
		return temp;
	}
	List<bool>* getCode()//geting binary code of elements
	{
		return cur->code;
	}
	void CreateCodes(Node* top, int count);//creating binary codes of symbols
	T getData();//geting data of elements
	void qsortRecursive(Node* arr, int size, int left_border); //quick sort
                                                };
```

List.cpp

```
#include "List.h"

template<typename T> bool List<T>::HasNext()
{
	if (this->Size != 0 && cur != nullptr)
		return true;
	else
		return false;
}
```

```
template<typename T> void List<T>::iterator(Node* curent)
{
	cur = curent;
}

template<typename T> void List<T>::push_with_count(T obj, int count)
{
	if (head != NULL) {
		this->tail->Next = new Node(obj, count);
		tail = tail->Next;
	}
	else {
		this->head = new Node(obj, count);
		this->tail = this->head;
	}
	Size++;
}

template<typename T> void List<T>::push_back(T obj) { // add to back of list
	if (head != NULL) {
		this->tail->Next = new Node(obj);
		tail = tail->Next;
	}
	else {
		this->head = new Node(obj);
		this->tail = this->head;
	}
	Size++;
}
template<typename T> void List<T>::push_front(T obj) { // add to head of list
	if (head != NULL) {
		Node* current = new Node;
		current->data = obj;
		current->Next = this->head;
		this->head = current;
	}
	else {
		this->head = new Node(obj);
	}
	this->Size++;
}
template<typename T> void List<T>::pop_back() { // delete last elem
	if (head != NULL) {
		Node* current = head;
		while (current->Next != tail)//search last
			current = current->Next;
		delete tail;
		tail = current;
		tail->Next = NULL;
		Size--;
	}
	else throw std::out_of_range("out_of_range");
}
template<typename T> void List<T>::pop_front() { // delete first elem
	if (head != NULL) {
		Node* current = head;
		head = head->Next;
		delete current;
		Size--;
	}
	else throw std::out_of_range("out_of_range");
}
template<typename T> void List<T>::insert(T obj, size_t k) {
```

```
	// add elem by index, insert before elem with this index
	if (k >= 0 && this->Size > k) {
		if (this->head != NULL) {
			if (k == 0)
				this->push_front(obj);
			else
				if (k == this->Size - 1)
					this->push_back(obj);
				else
				{
					Node* current = new Node;//for add elem
					Node* current1 = head;//for search result elem
					for (int i = 0; i < k - 1; i++) {
						current1 = current1->Next;
					}
					current->data = obj;
					current->Next = current1->Next;//change next elem
					current1->Next = current;
					Size++;
				}
		}
	}
	else {
		throw std::out_of_range("out_of_range");
	}
}
template<typename T> T List<T>::at(size_t k) {//get elem by index
	if (this->head != NULL && k >= 0 && k <= this->Size - 1) {
		if (k == 0)
			return this->head->data;
		else
			if (k == this->Size - 1)
				return this->tail->data;
			else
			{
				Node* current = head;
				for (int i = 0; i < k; i++) {
					current = current->Next;
				}
				return current->data;
			}
	}
	else {
		throw std::out_of_range("out_of_range");
	}
}

template<typename T> int List<T>::atCount(size_t k) {//get elem by index
	if (this->head != NULL && k >= 0 && k <= this->Size - 1) {
		if (k == 0)
			return this->head->count;
		else
			if (k == this->Size - 1)
				return this->tail->count;
			else
			{
				Node* current = head;
				for (int i = 0; i < k; i++) {
					current = current->Next;
				}
				return current->count;
			}
	}
	else {
```

```
		throw std::out_of_range("out_of_range");
	}
}

template<typename T> void List<T>::swap(size_t index_one, size_t index_two)
{
	T temp_info = this->at(index_one);
	int temp_count = this->atCount(index_one);
	this->set_with_count(index_one, this->at(index_two), this->atCount(index_two));
	this->set_with_count(index_two, temp_info, temp_count);
}
template<typename T> void List<T>::remove(int k) { // delete by index
	if (head != NULL && k >= 0 && k <= Size - 1) {
		if (k == 0) this->pop_front();
		else
			if (k == this->Size - 1) this->pop_back();
			else
				if (k != 0) {
					Node* current = head;
					for (int i = 0; i < k - 1; i++) {//go to before elem
						current = current->Next;
					}

					Node* current1 = current->Next;
					current->Next = current->Next->Next;
					delete current1;
					Size--;
				}
	}
	else {
		throw std::out_of_range("out_of_range");
	}
}
template<typename T> size_t List<T>::get_size() { // get list size
	return Size;
}
template<typename T> void List<T>::print_to_console() { //print elems to console without
delimetr
	if (this->head != NULL) {
		Node* current = head;
		for (int i = 0; i < Size; i++) {
			cout << current->data;
			current = current->Next;
		}
	}
}

template<typename T> void List<T>::print_to_console_with_code() {//print elems to console
with delimetr
	if (this->head != NULL) {
		Node* current = head;
		cout << endl;
		for (int i = 0; i < Size; i++) {
			cout <<"|"<< current->data << " - ";
			current->code->print_to_console();
			cout << endl;
			current = current->Next;
		}
	}
}
template<typename T> void List<T>::clear() { // clear list
	if (head != NULL) {
		Node* current = head;
		while (head != NULL) {
```

```
                        current = current->Next;
                        delete head;
                        head = current;
                }
                Size = 0;
        }
}
template<typename T> void List<T>::set(size_t k, T obj) { // change elem at index
        if (this->head != NULL && this->get_size() >= k && k >= 0) {
                Node* current = head;
                for (int i = 0; i < k; i++) {
                        current = current->Next;
                }
                current->data = obj;
        }
        else {
                throw std::out_of_range("out_of_range");
        }
}

template<typename T> void List<T>::set_with_count(size_t k, T obj, int count) {// change
elem at index
        if (this->head != NULL && this->get_size() >= k && k >= 0) {
                Node* current = head;
                for (int i = 0; i < k; i++) {
                        current = current->Next;
                }
                current->data = obj;
                current->count = count;
        }
        else {
                throw std::out_of_range("out_of_range");
        }
}

template<typename T> bool List<T>::isEmpty() { // check empty
        return (bool)(head);
}
template<typename T> void List<T>::reverse() { // reverse list
        int Counter = Size;
        Node* HeadCur = NULL;
        Node* TailCur = NULL;
        for (int j = 0; j < Size; j++) {
                if (HeadCur != NULL) {
                        if (head != NULL && head->Next == NULL) {
                                TailCur->Next = head;
                                TailCur = head;
                                head = NULL;
                        }
                        else {
                                Node* cur = head;
                                for (int i = 0; i < Counter - 2; i++)
                                        cur = cur->Next;
                                TailCur->Next = cur->Next;
                                TailCur = cur->Next;
                                cur->Next = NULL;
                                tail = cur;
                                Counter--;
                        }
                }
                else {
                        HeadCur = tail;
                        TailCur = tail;
                        Node* cur = head;
```

```
                        for (int i = 0; i < Size - 2; i++)
                                cur = cur->Next;
                        tail = cur;
                        tail->Next = NULL;
                        Counter--;
                }
        }
        head = HeadCur;
        tail = TailCur;
}

template<typename T> void List<T>::CreateCodes(Node* top, int count) {
        this->iterator(top);
        int sum = 0;
        int temp_count = 0;
        while (this->HasNext() && temp_count < count)
        {
                sum += this->cur->count;
                this->Next();
                temp_count++;
        }
        this->iterator(top);
        if (temp_count > 1) {
                int half = sum / 2;
                int halfSum = 0;
                int halfcount = 0;
                while (halfSum < half)
                {
                        this->cur->code->push_back(0);
                        halfSum += this->cur->count;
                        this->Next();
                        halfcount++;
                }
                if (halfcount > 1) {
                        this->iterator(top);
                        CreateCodes(cur, halfcount);
                }
                temp_count = halfcount;
                while (halfcount < count)
                {
                        this->cur->code->push_back(1);
                        halfSum += this->cur->count;
                        this->Next();
                        halfcount++;
                }
                if (halfcount - temp_count > 1) {
                        this->iterator(top);
                        for (int i = 0; i < temp_count; i++)
                        {
                                this->Next();
                        }
                        CreateCodes(cur, halfcount - temp_count);
                }
        }
        else if (temp_count == 1) {
                if (this->cur != NULL) {
                        this->cur->code = new List<bool>();
                        this->cur->code->push_back(0);
                }
        }

}
```

```
List<bool>* List<bool>::getCode()
{
	return cur->code;
}

template<typename T> T List<T>::getData() //get data of list
{
	if (cur != nullptr) return this->cur->data;
	else return NULL;
}

template<typename T> void List<T>::qsortRecursive(Node* arr, int size,int left_border) {
//sort list
	if (arr == nullptr)
	{
		return;
	}
	this->iterator(arr);
	//indexes of head and back of array
	int i =  left_border;
	int j = size - 1;

	//central elem

	int mid = this->atCount(size/2);
	//divide array
	do {
		//go through elems, found elems which should be remove to another part
		//in left part skip elems which less than middle
		while (this->atCount(i) < mid) {
			i++;
		}
		//in right part skip elems which higher than middle
		while (this->atCount(j) > mid) {
			j--;
		}
		//swap elems
		if (i <= j) {
			swap(i, j);
			i++;
			j--;
		}
	} while (i <= j);


	//reqursive
	if (j > 0) {
		qsortRecursive(arr, j + 1,left_border);
	}
	if (i < size) {
		qsortRecursive(arr, size - i,left_border);
	}
}
```

ShenonList.h

```
#include "List.cpp"


class ShenonList {
private:
	List<char>* word;
	List<char>* symbols;
```

```
	List<bool>* encodedWord;
	List<char>* decodedWord;
public:
	void input(); //input string,that need to encode
	void FindSymbols();//finding all symbols in out string
	void ListSort();//sort symbols in descending order
	void coding();//encoding our string
	void decoding();//decoding our string
	void print(); //print all info
	void setWord(List<char>* word);
	List<char>* getSymbols();
	void setSymbols(List<char>* symbols);
	List<bool>* get_encodedWord();
	List<char>* get_decodedWord();
	ShenonList()
	{
		this->word = new List<char>();
		this->symbols = new List<char>();
		this->encodedWord = new List<bool>();
		this->decodedWord = new List<char>();
	}
	~ShenonList()
	{
		this->word->clear();
		this->symbols->clear();
		this->encodedWord->clear();
		this->decodedWord->clear();
	}
};
```

ShenonList.cpp

```
#include "ShenonList.h"

void ShenonList::input()
{
	char s = 1;
	for (s = getchar(); s != '\n'; s = getchar())
	{
		word->push_back(s);
	}
	FindSymbols();
	ListSort();
	symbols->CreateCodes(symbols->getHead(), symbols->get_size());
	coding();
	decoding();
}
void ShenonList::FindSymbols()
{
	if (word->get_size() != 0) {
		int CountMass[256] = { 0 };
		word->iterator(word->getHead());
		while (word->HasNext())
		{
			CountMass[word->Next()->data]++;
		}
		for (int i = 0; i < 256; i++)
		{
			if (CountMass[i] > 0) {
				symbols->push_with_count(i, CountMass[i]);
			}
		}
	}
}
void ShenonList::ListSort()
```

```
{
	symbols->qsortRecursive(symbols->getHead(), symbols->get_size(), 0);
	symbols->reverse();
}


void ShenonList::coding()
{
	List<bool>* temp;
	this->word->iterator(this->word->getHead());
	this->symbols->iterator(this->symbols->getHead());
	while (this->word->HasNext())
	{
		while (this->symbols->getData() != this->word->getData())
		{
			this->symbols->Next();
		}
		temp = this->symbols->getCode();
		temp->iterator(temp->getHead());
		while (temp->HasNext())
		{
			this->encodedWord->push_back(temp->Next()->data);
		}
		this->symbols->iterator(this->symbols->getHead());
		this->word->Next();
	}
}

void ShenonList::decoding()
{
	List<bool>* temp;
	int index = 0;
	int match_index = 0;
	this->encodedWord->iterator(this->encodedWord->getHead());
	this->symbols->iterator(this->symbols->getHead());
	while (match_index != this->encodedWord->get_size())
	{
		temp = this->symbols->getCode();
		temp->iterator(temp->getHead());
		bool flag = false;
		while (temp->HasNext() && this->encodedWord->HasNext())
		{
			if (this->encodedWord->getData() == temp->getData())
				flag = true;
			else
			{
				flag = false;
				break;
			}
			temp->Next();
			this->encodedWord->Next();
			index++;
		}
		if (flag == true)
		{
			this->decodedWord->push_back(this->symbols->getData());
			match_index = index;
			this->symbols->iterator(this->symbols->getHead());
		}
		else {
			index = match_index;
			if (match_index != this->encodedWord->get_size()) {
				this->encodedWord->iterator(this->encodedWord->getHead());
				for (int i = 0; i < match_index; i++)
```

```
                                this->encodedWord->Next();

                        }
                        this->symbols->Next();
                        if (this->symbols->HasNext() == false)
                        {
                                this->symbols->iterator(this->symbols->getHead());
                        }
                }
        }
}

void ShenonList::print()
{
        cout << "Entered word - ";
        this->word->print_to_console();
        cout << endl;
        cout << "Table of codes";
        this->symbols->print_to_console_with_code();
        cout << endl;
        cout << "Encoded word - ";
        this->encodedWord->print_to_console();
        cout << endl;
        cout << "Decoded word - ";
        this->decodedWord->print_to_console();
        cout << endl;
        float compression_ratio;
        compression_ratio = (this->decodedWord->get_size() * 8.) / (this->encodedWord-
>get_size() * 1.);
        cout << "compression ratio is equal " << compression_ratio;
}

void ShenonList::setWord(List<char>* word)
{
        this->word = word;
}

List<char>* ShenonList::getSymbols()
{
        return this->symbols;
}
void ShenonList::setSymbols(List<char>* symbols)
{
        this->symbols = symbols;
}

List<bool>* ShenonList::get_encodedWord()
{
        return this->encodedWord;
}

List<char>* ShenonList::get_decodedWord()
{
        return this->decodedWord;
}
```

ShenonTest.cpp

```
#include "pch.h"
#include "CppUnitTest.h"
#include "../AISDLAB2b/ShenonList.cpp"
using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace ShenonTest
{
```

```
	TEST_CLASS(ShenonTest)
	{
	public:

		TEST_METHOD(CodingTest)
		{
			List<char>* word;
			word = new List<char>();
			word->push_back('z');
			word->push_back('x');
			ShenonList string;
			string.setWord(word);
			string.FindSymbols();
			string.ListSort();
			List<char>* symbols;
			symbols = new List<char>();
			symbols = string.getSymbols();
			symbols->CreateCodes(symbols->getHead(), symbols->get_size());
			string.setSymbols(symbols);
			string.coding();
			List<bool>* encoded_word;
			encoded_word = new List<bool>();
			encoded_word = string.get_encodedWord();
			Assert::AreEqual(encoded_word->at(1), false);
			Assert::AreEqual(encoded_word->at(0), true);
		}


		TEST_METHOD(CodingTest2)
		{
			List<char>* word;
			word = new List<char>();
			word->push_back('x');
			word->push_back('x');
			word->push_back('z');
			word->push_back('x');
			word->push_back('5');
			word->push_back('5');
			word->push_back('5');
			word->push_back('5');
			ShenonList string;
			string.setWord(word);
			string.FindSymbols();
			string.ListSort();
			List<char>* symbols;
			symbols = new List<char>();
			symbols = string.getSymbols();
			symbols->CreateCodes(symbols->getHead(), symbols->get_size());
			string.setSymbols(symbols);
			string.coding();
			List<bool>* encoded_word;
			encoded_word = new List<bool>();
			encoded_word = string.get_encodedWord();
			Assert::AreEqual(encoded_word->at(0), true);
			Assert::AreEqual(encoded_word->at(1), false);
			Assert::AreEqual(encoded_word->at(2), true);
			Assert::AreEqual(encoded_word->at(3), false);
			Assert::AreEqual(encoded_word->at(4), true);
			Assert::AreEqual(encoded_word->at(5), true);
			Assert::AreEqual(encoded_word->at(6), true);
			Assert::AreEqual(encoded_word->at(7), false);
			Assert::AreEqual(encoded_word->at(8), false);
			Assert::AreEqual(encoded_word->at(9), false);
			Assert::AreEqual(encoded_word->at(10), false);
```

```
		Assert::AreEqual(encoded_word->at(11), false);
	}

	TEST_METHOD(CodingTest3)
	{
		List<char>* word;
		word = new List<char>();
		word->push_back('x');
		word->push_back('x');
		word->push_back('x');
		word->push_back('x');
		word->push_back('x');
		word->push_back('x');
		word->push_back('x');
		word->push_back('x');
		ShenonList string;
		string.setWord(word);
		string.FindSymbols();
		string.ListSort();
		List<char>* symbols;
		symbols = new List<char>();
		symbols = string.getSymbols();
		symbols->CreateCodes(symbols->getHead(), symbols->get_size());
		string.setSymbols(symbols);
		string.coding();
		List<bool>* encoded_word;
		encoded_word = new List<bool>();
		encoded_word = string.get_encodedWord();
		Assert::AreEqual(encoded_word->at(0), false);
		Assert::AreEqual(encoded_word->at(1), false);
		Assert::AreEqual(encoded_word->at(2), false);
		Assert::AreEqual(encoded_word->at(3), false);
		Assert::AreEqual(encoded_word->at(4), false);
		Assert::AreEqual(encoded_word->at(5), false);
		Assert::AreEqual(encoded_word->at(6), false);
		Assert::AreEqual(encoded_word->at(7), false);
	}

	TEST_METHOD(CodingTest4)
	{
		List<char>* word;
		word = new List<char>();
		word->push_back('x');
		word->push_back('z');
		word->push_back('x');
		word->push_back('z');
		word->push_back('x');
		word->push_back('z');
		word->push_back('x');
		word->push_back('z');
		ShenonList string;
		string.setWord(word);
		string.FindSymbols();
		string.ListSort();
		List<char>* symbols;
		symbols = new List<char>();
		symbols = string.getSymbols();
		symbols->CreateCodes(symbols->getHead(), symbols->get_size());
		string.setSymbols(symbols);
		string.coding();
		List<bool>* encoded_word;
		encoded_word = new List<bool>();
		encoded_word = string.get_encodedWord();
		Assert::AreEqual(encoded_word->at(0), false);
```

```
            Assert::AreEqual(encoded_word->at(1), true);
            Assert::AreEqual(encoded_word->at(2), false);
            Assert::AreEqual(encoded_word->at(3), true);
            Assert::AreEqual(encoded_word->at(4), false);
            Assert::AreEqual(encoded_word->at(5), true);
            Assert::AreEqual(encoded_word->at(6), false);
            Assert::AreEqual(encoded_word->at(7), true);
        }


        TEST_METHOD(DecodingTest)
        {
            List<char>* word;
            word = new List<char>();
            word->push_back('x');
            word->push_back('x');
            word->push_back('z');
            word->push_back('x');
            word->push_back('5');
            word->push_back('5');
            word->push_back('5');
            word->push_back('5');
            ShenonList string;
            string.setWord(word);
            string.FindSymbols();
            string.ListSort();
            List<char>* symbols;
            symbols = new List<char>();
            symbols = string.getSymbols();
            symbols->CreateCodes(symbols->getHead(), symbols->get_size());
            string.setSymbols(symbols);
            string.coding();
            string.decoding();
            List<char>* decoded_word;
            decoded_word = new List<char>();
            decoded_word = string.get_decodedWord();
            Assert::AreEqual(decoded_word->at(0), 'x');
            Assert::AreEqual(decoded_word->at(1), 'x');
            Assert::AreEqual(decoded_word->at(2), 'z');
            Assert::AreEqual(decoded_word->at(3), 'x');
            Assert::AreEqual(decoded_word->at(4), '5');
            Assert::AreEqual(decoded_word->at(5), '5');
            Assert::AreEqual(decoded_word->at(6), '5');
            Assert::AreEqual(decoded_word->at(7), '5');
        }

        TEST_METHOD(DecodingTest2)
        {
            List<char>* word;
            word = new List<char>();
            word->push_back('z');
            word->push_back('x');
            ShenonList string;
            string.setWord(word);
            string.FindSymbols();
            string.ListSort();
            List<char>* symbols;
            symbols = new List<char>();
            symbols = string.getSymbols();
            symbols->CreateCodes(symbols->getHead(), symbols->get_size());
            string.setSymbols(symbols);
            string.coding();
            string.decoding();
            List<char>* decoded_word;
```

```
                decoded_word = new List<char>();
                decoded_word = string.get_decodedWord();
                Assert::AreEqual(decoded_word->at(0), 'z');
                Assert::AreEqual(decoded_word->at(1), 'x');
        }

        TEST_METHOD(FindingTest)
        {
                List<char>* word;
                word = new List<char>();
                word->push_back('z');
                word->push_back('x');
                ShenonList string;
                string.setWord(word);
                string.FindSymbols();
                string.ListSort();
                List<char>* symbols;
                symbols = new List<char>();
                symbols = string.getSymbols();
                Assert::AreEqual(symbols->at(0), 'x');
                Assert::AreEqual(symbols->at(1), 'z');
        }

        TEST_METHOD(FindingTest2)
        {
                List<char>* word;
                word = new List<char>();
                word->push_back('x');
                word->push_back('x');
                word->push_back('z');
                word->push_back('x');
                word->push_back('5');
                word->push_back('5');
                word->push_back('5');
                word->push_back('5');
                ShenonList string;
                string.setWord(word);
                string.FindSymbols();
                string.ListSort();
                List<char>* symbols;
                symbols = new List<char>();
                symbols = string.getSymbols();
                Assert::AreEqual(symbols->at(0), '5');
                Assert::AreEqual(symbols->at(1), 'x');
                Assert::AreEqual(symbols->at(2), 'z');
        }
    };
}
```

Выводы

Реализовал алгоритм кодирования Шеннона-Фано в C++.