

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования «Тульский государственный
университет»

КАФЕДРА ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

ЛИНЕЙНЫЕ СПИСКИ

отчет о
лабораторной работе №8

по дисциплине
ТЕХНОЛОГИИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ

ВАРИАНТ 6

Выполнила:	ст. гр. 230711	Павлова В.С.
Проверил:	асс. каф. ИБ	Курбаков М.Ю.

Тула, 2022 г.

ЦЕЛЬ И ЗАДАЧА РАБОТЫ

Цель: ознакомиться с понятием линейных списков, основными видами линейных списков, примерами их применения, научиться описывать и использовать линейные списки.

Задача: в данной работе требуется написать программу, демонстрирующую использование изученных принципов.

ЗАДАНИЕ НА РАБОТУ

Из последовательности символов, состоящей из n элементов и организованной как линейный список, получать последовательность вида $c_n, c_{n-1}, \dots, c_1, c_2, \dots, c_m$, где $m < n$.

СХЕМА ПРОГРАММЫ

Схема алгоритма программы представлена на рисунке 1.

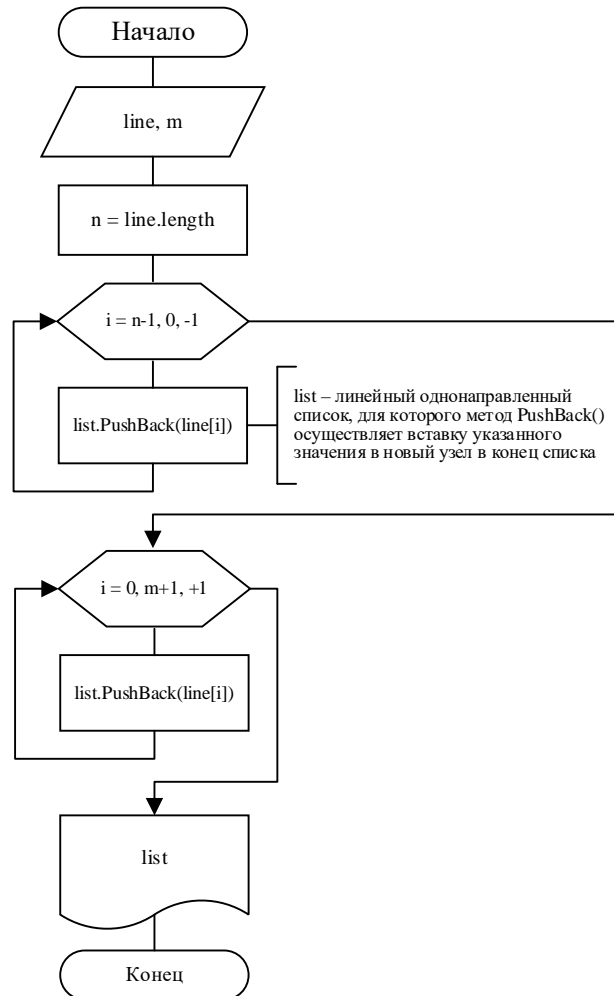


Рисунок 1 – Схема алгоритма программы

ТЕКСТ ПРОГРАММЫ

Текст программы на языке программирования C++ для решения задания по варианту представлен в листингах 1 и 2. В листинге 1 представлено содержимое файла main.cpp, в листинге 2 – содержимое заголовочного файла LinkedList.h, описывающего класс линейного списка.

Листинг 1. Текст программы (содержимое файла main.cpp)

```
#include <iostream>
#include <string>
#include <Windows.h>
#include "LinkedList.h"

int main()
{
    setlocale(LC_ALL, "RUSSIAN");
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    LinkedList <char> list;
    string line;
    cout << "\n\tВведите последовательность символов для добавления в список:
";
    getline(cin >> ws, line);
    int n = line.length();
    for (size_t i = n-1; i > 0 ; i--)
        list.PushBack(line[i]);
    int m;
    cout << "\n\tВведите параметр m (0 <= m < " << n << "): "; cin >> m;
    for (size_t i = 0; i <= m; i++)
        list.PushBack(line[i]);
    list.PrintList();
    cout << "\n";
    return 0;
}
```

Листинг 2. Текст программы (содержимое файла LinkedList.h)

```
#pragma once
#include <iostream>
using namespace std;

template <typename T>
struct Node //структура узла списка
{
    T value;
    Node<T>* next;

    Node(T data, Node<T>* nextPtr = nullptr)
    {
        value = data;
        next = nextPtr;
    }
    ~Node(){}
};

template <typename T>
class LinkedList
{
public:
    Node<T>* head;
    int size;

    LinkedList()
    {
        head = nullptr;
        size = 0;
    }

    LinkedList(const LinkedList<T>& origList) //конструктор копий
    {
        head = nullptr;
    }
};
```

Листинг 2. Текст программы (продолжение)

```
        size = 0;
        Node<T>* origPtr = origList.head;
        for (size_t i = 0; i < origList.size; i++)
        {
            (*this).PushBack(origPtr->value);
            origPtr = origPtr->next;
        }
    }

    void PrintList() //вывести содержимое всех узлов
    {
        Node<T>* curPtr = head;
        cout << "\n\t";
        for (int i = 0; i < size; i++)
        {
            cout << curPtr->value << " ";
            curPtr = curPtr->next;
        }
    }

    void PushBack(const Node<T>* &node) //добавить узел node в конец списка
    {
        Node<T>* newNode(node->value);
        if (head == nullptr)
        {
            head = newNode;
            size++;
            return true;
        }
        GetNode(size - 1)->next = newNode;
        size++;
        return true;
    }

    void PushBack(const T &data) //добавить новый узел со значением
    { //data в конец списка
        T newData = data;
        Node<T>* node = new Node<T>(newData);
        if (head == nullptr)
        {
            head = node;
            size++;
            return;
        }
        GetNode(size - 1)->next = node;
        size++;
    }

    ~LinkedList() //деструктор
    {
        Node<T>* ptr = head;
        Node<T>* next = nullptr;
        for (size_t i = 0; i < size; i++)
        {
            next = head->next;
            delete head;
            head = next;
        }
        size = 0;
    }
};
```

ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ

Данная программа предназначена для того, чтобы из последовательности символов, состоящей из n элементов и организованной как линейный список, получать последовательность вида $c_n, c_{n-1}, \dots, c_1, c_2, \dots, c_m$, где $m < n$. Пользователю предлагается ввести входную последовательность символов, а также размер параметра m .

ИНСТРУКЦИЯ ПРОГРАММИСТА

Структуры данных, используемые в программе, приведены в таблице 1. Описание класса `LinkedList` приведено в таблице 2.

Таблица 1 – Структуры данных в программе

Имя	Тип (класс)	Предназначение
<code>n</code>	<code>int</code>	Длина входной последовательности
<code>m</code>	<code>int</code>	Вводимый параметр
<code>list</code>	<code>LinkedList</code>	Линейный однонаправленный список
<code>line</code>	<code>string</code>	Входная последовательность

Таблица 2 – Описание разработанного класса

<code>class LinkedList</code>	
Поля/свойства (элементы данных) класса	
Название и тип	Описание
<code>Node<T>* head</code>	Корень списка
<code>int size</code>	Количество узлов в списке
Шаблон для описания типа данных, хранящихся в узлах списка	
<code>template <typename T></code>	

Таблица 2 – Описание разработанного класса (продолжение)

Методы (функции-элементы) класса		
Название и тип возвращаемого значения	Аргументы	Описание
LinkedList()	Node<T>*head = nullptr, int size = 0	Конструктор класса
LinkedList()	const LinkedList<T>& origList	Конструктор копий
bool PushBack()	T element	Добавить узел в конец списка, в случае успеха вернёт true
void PrintList()	void	Вывод списка всех узлов и хранящихся в них значений
~List()	void	Деструктор класса

Для реализации данного класса была создана вспомогательная структура Node (узел), описание которой представлено в таблице 3.

Таблица 3 – Описание разработанной структуры Node

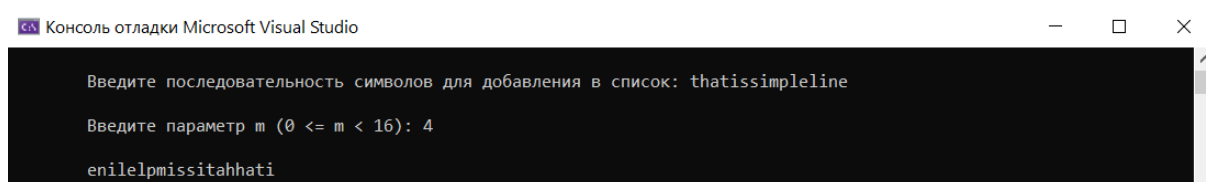
struct Node		
Поля/свойства (элементы данных) структуры		
Название и тип	Описание	
T value	Значение, которое хранится в узле	
Node<T>* next	Ссылка на следующий элемент списка	
Шаблон для описания типа данных, хранящихся в узлах списка		
template <typename T>		
Методы (функции-элементы) структуры		
Название и тип возвращаемого значения	Аргументы	Описание
Node ()	T data Node<T>* nextPtr=nullptr	Конструктор структуры
~Node ()	void	Деструктор структуры

ДЕМОНСТРАЦИОННЫЙ ПРИМЕР

Рассмотрим принцип действия программы на конкретном примере. Данная программа должна из входной последовательности символов, состоящей из n элементов и организованной как линейный список, получать последовательность вида $c_n, c_{n-1}, \dots, c_1, c_2, \dots, c_m$, где $m < n$.

Пусть входная последовательность имеет вид: «thatissimpleline», её длина $n = 16$ символов, а величину m возьмём равной 4. Чтобы получить искомую последовательность вида $c_n, c_{n-1}, \dots, c_1, c_2, \dots, c_m$, необходимо записать входную зеркально, то есть в обратном порядке, а после в конец приписать ещё 4 символа из начала входной строки, начиная со второго (символ c_1 не дублируется, после него сразу идёт c_2).

Тогда после преобразований последовательность должна иметь следующий вид: «enilelpmissitahthati». Результат работы программы (рисунок 2) для данного набора входных данных соответствует полученному теоретически.



```
Консоль отладки Microsoft Visual Studio

Введите последовательность символов для добавления в список: thatissimpleline

Введите параметр m (0 <= m < 16): 4

enilelpmissitahthati
```

Рисунок 2 – Результат работы программы

ВЫВОДЫ

В ходе данной лабораторной работы был изучен принцип работы с линейными списками, которые представляют собой элементы данных со структурными взаимосвязями. Для демонстрации полученных знаний была написана программа, оперирующая входными данными как линейным списком. По результатам проверки программы можно сделать вывод о том, что она работает корректно.