Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №11.1**

Дисциплина: основы алгоритмизации и программирования

Тема: Однонаправленные списки в С++

Вариант 24

Выполнил работу

студент группы РИС-20-1б

Нечаев Д.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2020

**Цель задачи**

Цель – написать программу для создания, редактирования и вывода однонаправленного списка.

Для достижения поставленной цели необходимо решить следующие задачи:

* Проанализировать логику однонаправленного списка
* Продумать, как будет создан однонаправленный список в с++
* Создать класс, в котором будет создан первый элемент списка и основные методы для обработки однонаправленного списка
* Создать основные методы для обработки однонаправленного списка
* Написать алгоритм, который будет редактировать исходный список

Постановка задачи

Тип информационного поля char\*. Добавить в список элементы с номерами 1, 3, 5 и т. д.

Анализ задачи

1. Определить какие операции должны быть выполнены по заданию:

* Создание класса List в котором будут храниться методы и первый элемент списка
* Описать в public секции класса List основные методы для работы со списком:

class List

{

public:

List();

void push\_back(T data);

int GetSize() { return Size; }

T& operator[](const int index);

void insert(T value, int index);

* Описать в private секции класса List класс Node, который будет хранить узлы списка

private:

template <typename T>

class Node

{

public:

Node \*pNext;

T data;

Node(T data = T(), Node\* pNext = nullptr)

{

this->data = data;

this->pNext = pNext;

}

};

int Size;

Node <T> \*head;

* Создать шаблоны функций для корректной работы с различными типами данных
* Описать основные методы для обработки однонаправленного списка

1. Для решения задачи используются переменные:

Несколько целочисленных переменных для управления циклами и контроля размера списка

int numbersCount = -1, j, k;

while (numbersCount <= 0)

{

cout << "Введи количество элементов в списке: ";

cin >> numbersCount;

}

Указатели на следующие элементы списка

Node \*pNext;

Шаблонные переменные, в которых хранится значение узлов списка

T data;

1. Ввод данных осуществляется посредством функции

cin >> numbersCount;

Вывод данных осуществляется посредством функции

cout << i << ")" << lst[i] << " ";

1. Для решения задачи будут использованы циклы. Например, цикл, который создает некоторое количество новых элементов в списке и заносит в них данные:

for (int i = 0; i < numbersCount; i++)

{

cout << i << ")";

cin >> k;

lst.push\_back(k);

}

1. Для защиты программы от случайных ошибок были реализованы проверки входных данных:

Проверка размерности массива (размер больше 0):

int numbersCount = -1;

while (numbersCount <= 0)

{

cout << "Введи количество элементов в списке: ";

cin >> numbersCount;

}

1. Для описания основных действий над списком создаются методы класса List. Для того, чтобы избежать нагромождений кода, методы вынесены из класса. Например метод push\_back добавляет новый элемент в конец списка:

template<typename T>

void List<T>::push\_back(T data)

{

if (head == nullptr)

{

head = new Node <T>(data);

}

else

{

Node<T>\* current = this->head;

while (current->pNext != nullptr)

{

current = current->pNext;

}

current->pNext = new Node<T>(data);

}

Size++;

}

1. Для того, чтобы список мог корректно работать с данными различных основных типов, применен шаблон T

template <typename T>

В последствии все классы, подклассы и методы описаны с применением шаблонов

template<typename T>

void List<T>::insert(T value, int index)

{

Node<T>\* previous = this->head;

for (int i = 0; i < index - 1; i++)

{

previous = previous->pNext;

}

Node<T> \*newNode = new Node<T>(value, previous->pNext);

previous->pNext = newNode;

Size++;

}

Код

#include <iostream>

#include <string>

#include <list>

using namespace std;

template <typename T>

class List

{

public:

List();

void push\_back(T data);

int GetSize() { return Size; }

T& operator[](const int index);

void insert(T value, int index);

private:

template <typename T>

class Node

{

public:

Node \*pNext;

T data;

Node(T data = T(), Node\* pNext = nullptr)

{

this->data = data;

this->pNext = pNext;

}

};

int Size;

Node <T> \*head;

};

template <typename T>

List<T>::List()

{

Size = 0;

head = nullptr;

}

template<typename T>

void List<T>::push\_back(T data)

{

if (head == nullptr)

{

head = new Node <T>(data);

}

else

{

Node<T>\* current = this->head;

while (current->pNext != nullptr)

{

current = current->pNext;

}

current->pNext = new Node<T>(data);

}

Size++;

}

template<typename T>

T& List<T>::operator[](const int index)

{

int counter = 0;

Node<T>\* current = this->head;

while (current!=nullptr)

{

if (counter == index)

{

return current->data;

}

current = current->pNext;

counter++;

}

}

template<typename T>

void List<T>::insert(T value, int index)

{

Node<T>\* previous = this->head;

for (int i = 0; i < index - 1; i++)

{

previous = previous->pNext;

}

Node<T> \*newNode = new Node<T>(value, previous->pNext);

previous->pNext = newNode;

Size++;

}

int main()

{

List<char> lst;

char k;

int numbersCount = -1;

while (numbersCount <= 0)

{

cout << "Введи количество элементов в списке: ";

cin >> numbersCount;

}

cout << "Введи элементы списка:" << endl;

for (int i = 0; i < numbersCount; i++)

{

cout << i << ")";

cin >> k;

lst.push\_back(k);

}

cout << "Новый список:" << endl;

for (int i = 1; i < lst.GetSize(); i = i + 2)

{

lst.insert('A', i);

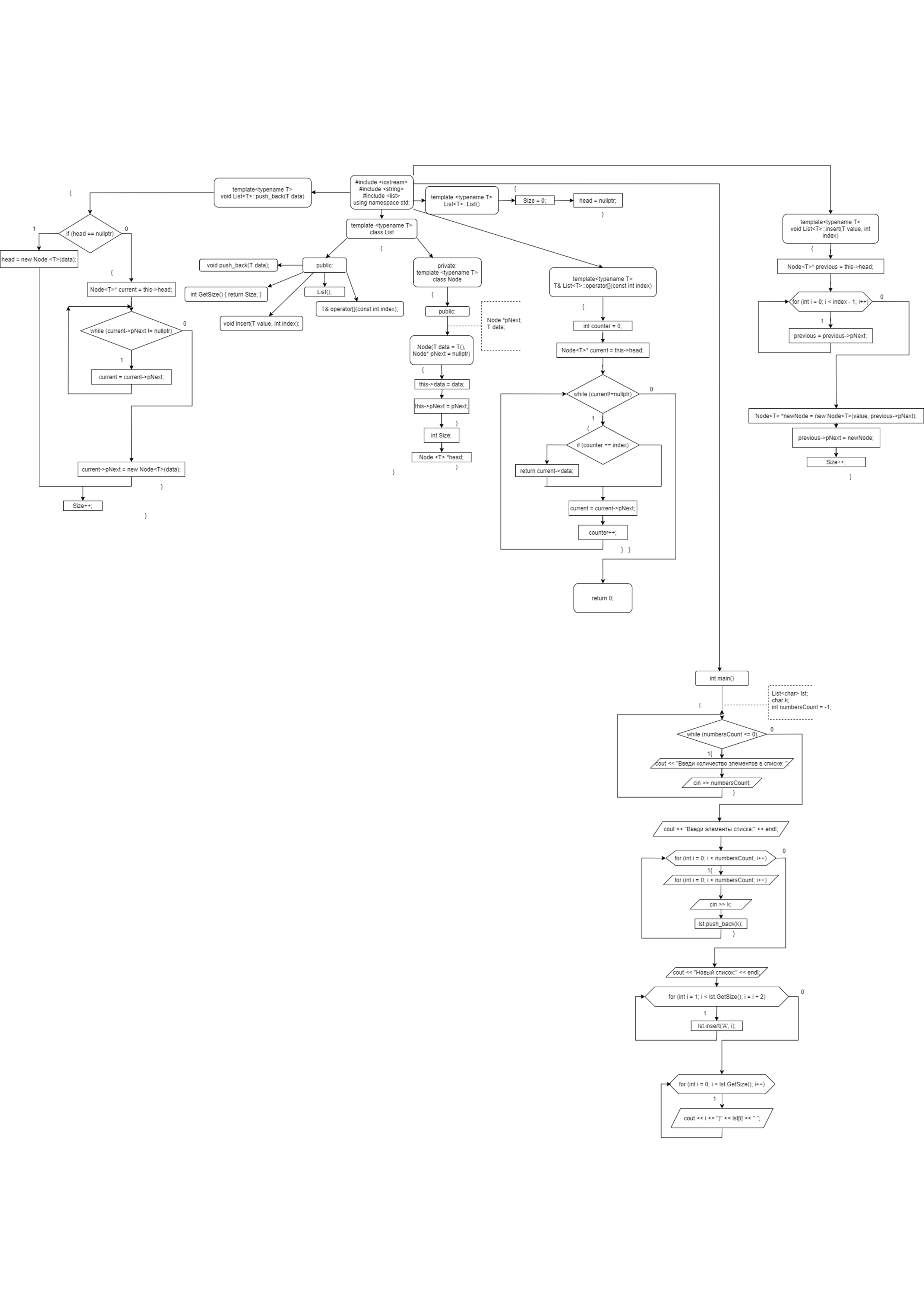
}

for (int i = 0; i < lst.GetSize(); i++)

cout << i << ")" << lst[i] << " ";

}

Блок-схема:



Работа кода

