Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа №11**

**«Создание двусвязного списка»**

**19 вариант**

Выполнил:

Студент группы РИС-23-1б

Шароглазов Егор Алексеевич

Проверила:

Доцент кафедры ИТАС

О.А. Полякова

2024 г

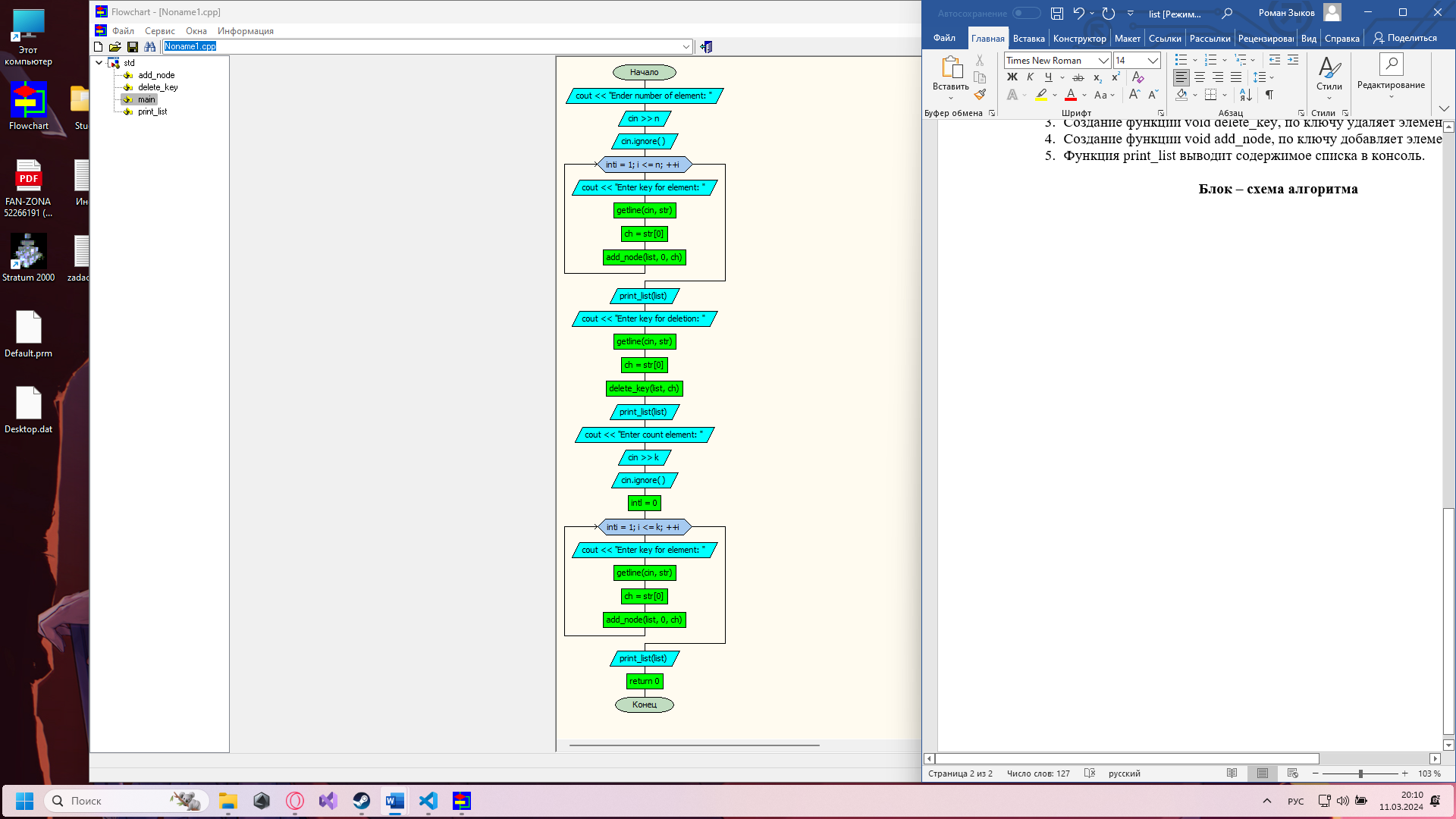
**Постановка задачи:** Записи в линейном списке содержат ключевое поле типа \*char(строка символов). Сформировать двунаправленный список. Удалить элемент с заданным ключом. Добавить К элементов в начало списка.

**Анализ задачи:**

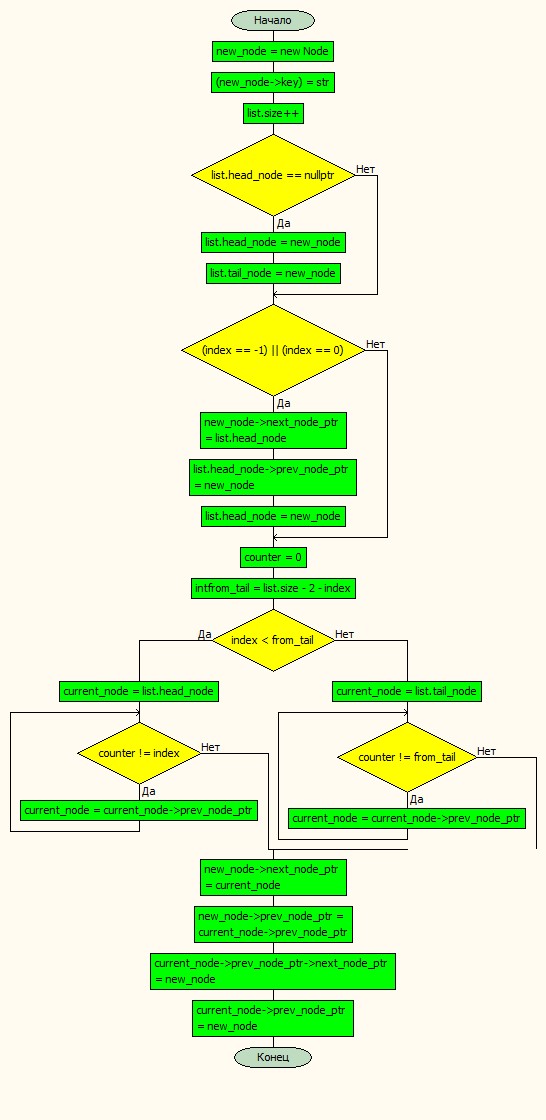
1. Создание структуры Node, создающую узел. Связывает узел указателями с предыдущим и следующим элементом.
2. Создание структуры List, head\_node и tail\_node указывают на nullptr.
3. Создание функции void delete\_key, по ключу удаляет элемент из списка.
4. Создание функции void add\_node, по ключу добавляет элемент в список.
5. Функция print\_list выводит содержимое списка в консоль.

**Блок – схема алгоритма**

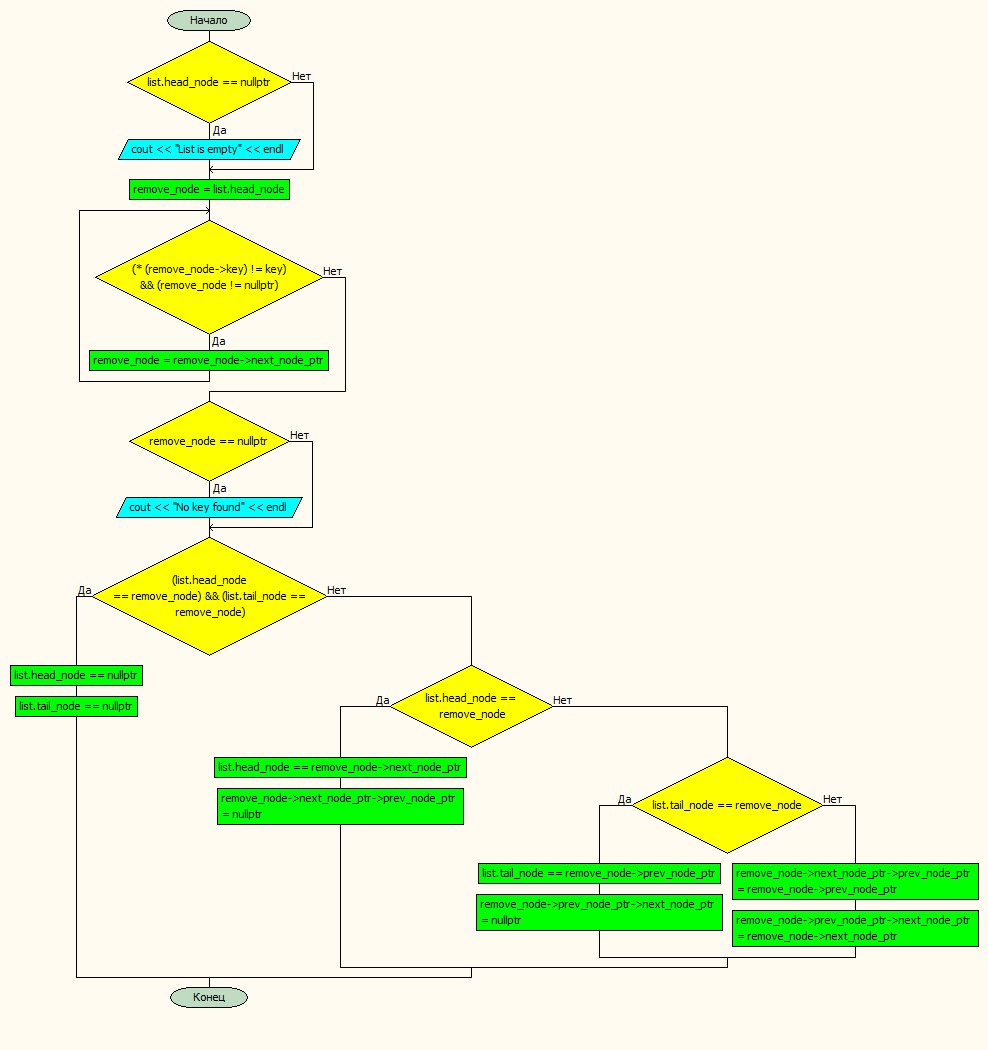
int main()



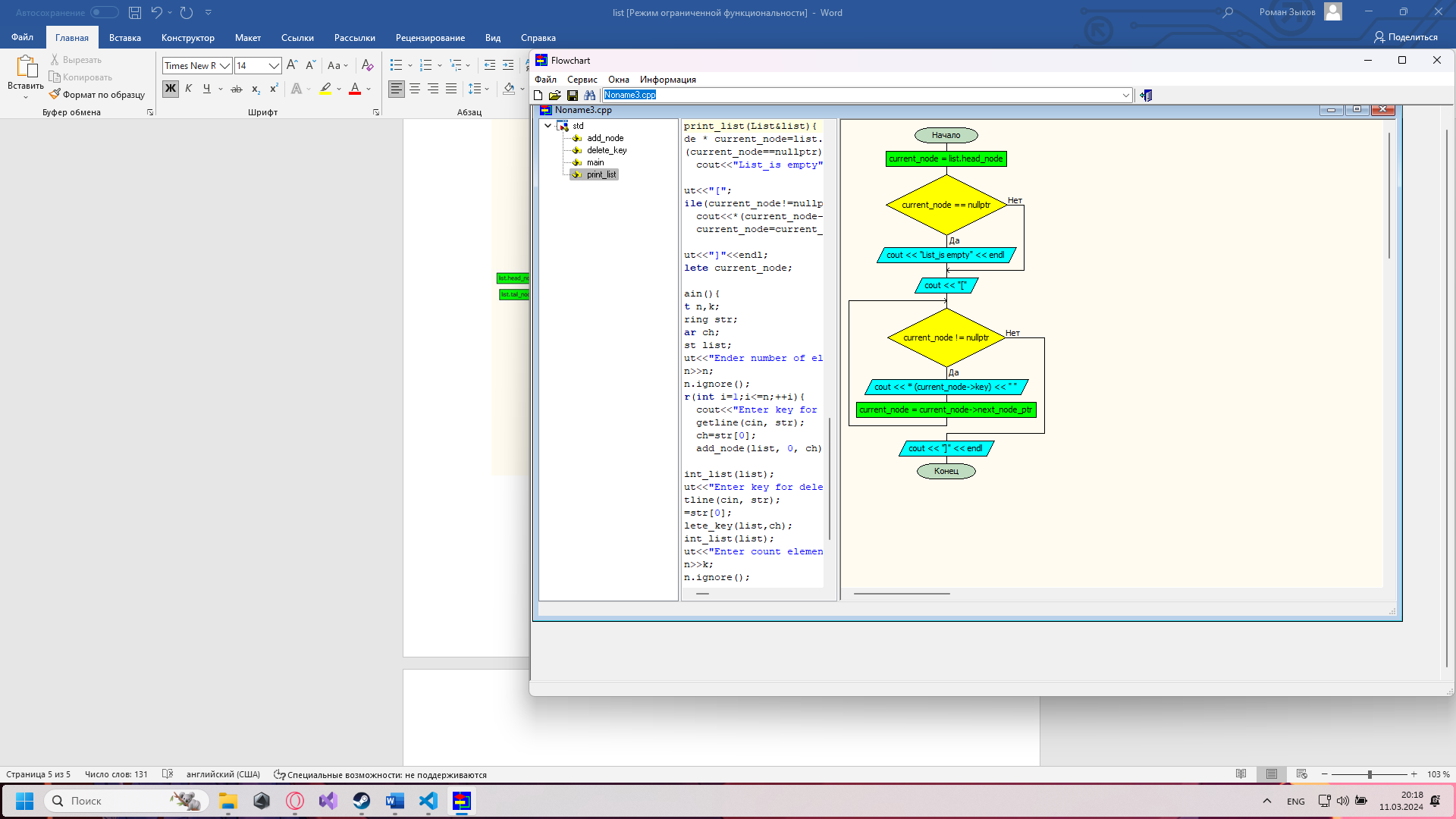
**add\_node()**



**delete\_key()**



**print\_list**



**Код программы**

#include <iostream>

#include <string.h>

#include <string>

using namespace std;

struct Node{

    char \*key=new char[1];

    Node \*prev\_node\_ptr=nullptr;

    Node \*next\_node\_ptr=nullptr;

};

struct List{

    Node\*head\_node=nullptr;

    Node\* tail\_node=nullptr;

    int size=0;

};

 void delete\_key(List& list,char key){

    if(list.head\_node==nullptr){

        cout<<"List is empty"<<endl;

        return;

    }

    Node\* remove\_node=list.head\_node;

    while((\*(remove\_node->key)!=key)&&(remove\_node!=nullptr)){

        remove\_node=remove\_node->next\_node\_ptr;

    }

    if(remove\_node==nullptr){

        cout<<"No key found"<<endl;

        return;

    }

    if((list.head\_node==remove\_node)&&(list.tail\_node==remove\_node)){

        list.head\_node==nullptr;

        list.tail\_node==nullptr;

    }

    else if(list.head\_node==remove\_node){

        list.head\_node==remove\_node->next\_node\_ptr;

        remove\_node->next\_node\_ptr->prev\_node\_ptr=nullptr;

    }

    else if(list.tail\_node==remove\_node){

        list.tail\_node==remove\_node->prev\_node\_ptr;

        remove\_node->prev\_node\_ptr->next\_node\_ptr=nullptr;

    }

    else{

        remove\_node->next\_node\_ptr->prev\_node\_ptr=remove\_node->prev\_node\_ptr;

        remove\_node->prev\_node\_ptr->next\_node\_ptr=remove\_node->next\_node\_ptr;

    }

    delete remove\_node;

 }

 void add\_node(List& list,int index,char str){

    Node \*new\_node=new Node;

    \*(new\_node->key)=str;

    list.size++;

    if(list.head\_node==nullptr){

        list.head\_node=new\_node;

        list.tail\_node=new\_node;

        return;

    }

    if((index==-1)||(index==0)){

        new\_node->next\_node\_ptr=list.head\_node;

        list.head\_node->prev\_node\_ptr=new\_node;

        list.head\_node=new\_node;

        return;

    }

    int counter=0;

    int from\_tail=list.size-2-index;

    Node\*current\_node;

    if(index<from\_tail){

        current\_node=list.head\_node;

        while(counter!=index){

            current\_node=current\_node->prev\_node\_ptr;

            ++counter;

        }

    }

    else{

        current\_node=list.tail\_node;

        while(counter!=from\_tail){

            current\_node=current\_node->prev\_node\_ptr;

            ++counter;

        }

    }

    new\_node->next\_node\_ptr=current\_node;

    new\_node->prev\_node\_ptr=current\_node->prev\_node\_ptr;

    current\_node->prev\_node\_ptr->next\_node\_ptr=new\_node;

    current\_node->prev\_node\_ptr=new\_node;

 }

 void print\_list(List&list){

    Node \* current\_node=list.head\_node;

    if(current\_node==nullptr){

        cout<<"List\_is empty"<<endl;

        return;

    }

    cout<<"[";

    while(current\_node!=nullptr){

        cout<<\*(current\_node->key)<<" ";

        current\_node=current\_node->next\_node\_ptr;

    }

    cout<<"]"<<endl;

    delete current\_node;

 }

 int main(){

    int n,k;

    string str;

    char ch;

    List list;

    cout<<"Ender number of element: ";

    cin>>n;

    cin.ignore();

    for(int i=1;i<=n;++i){

        cout<<"Enter key for element: ";

        getline(cin,str);

        ch=str[0];

        add\_node(list,0,ch);

    }

    print\_list(list);

    cout<<"Enter key for deletion: ";

    getline(cin,str);

    ch=str[0];

    delete\_key(list,ch);

    print\_list(list);

    cout<<"Enter count element: ";

    cin>>k;

    cin.ignore();

    int l=0;

    for(int i=1;i<=k;++i){

        cout<<"Enter key for element: ";

        getline(cin,str);

        ch=str[0];

        add\_node(list,0,ch);

    }

    print\_list(list);

    return 0;

 }

**Примеры работы программы**

