Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Информатика»

Тема: Лабораторная работа 11

"Динамические структуры данных"

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Протасов Н.С.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи**

Написать программу, в которой реализуется работа с односвязным списком.

Для каждого варианта разобрать следующие функции:

1. Удаление данных из списка

2. Печать списка

3. Удаление элемента в начале списка

4. Удаление элемента в конце списка

5. Добавление элемента в начало списка

6. Добавление элемента в конец списка

7.Добавление элемента после элемента с заданным номером

8. Удаление элемента по номеру

9. Добавление K элементов после элемента с заданным номером

10. Удаление k элементов после элемента с заданным номером

**Анализ задачи**

**Связный список, содержащий только один указатель на следующий элемент, называется односвязным.**

Для реализации поставленной задачи и функций, создадим структуры – Listи Node.

structNode {

int data;

Node\* next;

Node\* prev;

};

structList {

Node\* head;

Node\* tail;

int count;

};

**Программныйкод**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

struct List {

Node\* head;

int count;

};

void ShowList(List& list);

void PushFront(List& list, int value);

int PopFront(List& list, bool& flag);

void PushBack(List& list, int value);

int PopBack(List& list, bool& flag);

int AddAfter(List& list, int value, int number);

void RemoveElement(List& list, int number);

int AddSeveral(List& list, int number, int k);

void ClearList(List& list);

int main()

{

List list;

list.head = NULL;

list.count = 0;

int n, value, count, k;

bool flag;

do {

cout << "\n";

cout << "0. Exit\n";

cout << "1. Show List\n";

cout << "2. Add Element In Front Of List\n";

cout << "3. Add Element In End Of List\n";

cout << "4. Delete First Element\n";

cout << "5. Delete Last Element\n";

cout << "6. Add Element After k-Element\n";

cout << "7. Delete k-Element\n";

cout << "8. Add N Elements After k-Element\n";

cout << "9. Delete N Elements After k-Element\n";

cout << "10. Clear List\n";

cout << "\n";

cin >> n;

switch (n)

{

case 1:

{

ShowList(list);

break;

}

case 2:

{

cout << "Enter Value\n";

cin >> value;

PushFront(list, value);

break;

}

case 3:

{

cout << "Enter Value\n";

cin >> value;

PushBack(list, value);

break;

}

case 4:

{

value = PopFront(list, flag);

if (!flag)

{

cout << "List Is Empty\n";

}

else

{

cout << "Element " << value << " Has Been Removed\n";

}

break;

}

case 5:

{

value = PopBack(list, flag);

if (!flag) cout << "List Is Empty\n";

else cout << "Element " << value << " Has Been Removed\n";

break;

}

case 6:

{

cout << "Enter Index Of k-Element\n";

cin >> k;

cout << "Enter Value\n";

cin >> value;

value = AddAfter(list, value, k);

if (value == -1) cout << "Something Goes Wrong\n";

break;

}

case 7:

{

cout << "Enter Index Of k-Element\n";

cin >> k;

RemoveElement(list, k);

break;

}

case 8:

{

cout << "Enter Index Of k-Element\n";

cin >> k;

cout << "Count Of Elements\n";

cin >> count;

value = AddSeveral(list, k, count);

if (value == -1) cout << "Something Goes Wrong\n";

break;

}

case 9:

{

cout << "Enter Index Of k-Element\n";

cin >> k;

cout << "Count Of Elements\n";

cin >> count;

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

{

RemoveElement(list, k + 1);

}

break;

}

case 10:

{

ClearList(list);

cout << "List Cleared\n";

break;

}

}

} while (n != 0);

}

void ShowList(List& list) {

Node\* temp = list.head;

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

void PushFront(List& list, int value) {

Node\* newNode = new Node;

newNode->data = value;

newNode->next = list.head;

list.head = newNode;

list.count++;

}

int PopFront(List& list, bool& flag) {

if (list.head == nullptr) {

flag = false;

return -1;

}

flag = true;

int result = list.head->data;

Node\* temp = list.head;

list.head = list.head->next;

list.count--;

delete temp;

return result;

}

void PushBack(List& list, int value) {

Node\* newNode = new Node;

newNode->data = value;

newNode->next = nullptr;

if (list.head == nullptr) {

list.head = newNode;

}

else {

Node\* temp = list.head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

list.count++;

}

int PopBack(List& list, bool& flag) {

if (list.head == nullptr) {

flag = false;

return -1;

}

flag = true;

int result;

if (list.head->next == nullptr) {

result = list.head->data;

delete list.head;

list.head = nullptr;

}

else {

Node\* temp = list.head;

while (temp->next->next != nullptr) {

temp = temp->next;

}

result = temp->next->data;

delete temp->next;

temp->next = nullptr;

}

list.count--;

return result;

}

int AddAfter(List& list, int value, int number) {

if (list.head == nullptr || number >= list.count) {

return -1;

}

Node\* current = list.head;

for (int i = 1; i <= number && current != nullptr; i++) {

current = current->next;

}

if (current == nullptr) {

return -1;

}

Node\* newNode = new Node;

newNode->data = value;

newNode->next = current->next;

current->next = newNode;

current = newNode;

list.count++;

return 0;

}

void RemoveElement(List& list, int number) {

number++;

if (number < 1 || number > list.count) {

return;

}

if (number == 1) {

Node\* temp = list.head;

list.head = list.head->next;

delete temp;

list.count--;

return;

}

Node\* temp = list.head;

for (int i = 1; i < number - 1; i++) {

temp = temp->next;

}

Node\* delNode = temp->next;

temp->next = temp->next->next;

delete delNode;

list.count--;

}

int AddSeveral(List& list, int number, int k) {

if (list.head == nullptr || number >= list.count || k <= 0) {

return -1;

}

Node\* current = list.head;

for (int i = 1; i <= number && current != nullptr; i++) {

current = current->next;

}

if (current == nullptr) {

return -1;

}

for (int i = 0; i < k; i++) {

int value;

cout << "Enter Value\n";

cin >> value;

Node\* newNode = new Node;

newNode->data = value;

newNode->next = current->next;

current->next = newNode;

current = newNode;

list.count++;

}

return 0;

}

void ClearList(List& list) {

Node\* temp = list.head;

while (temp != nullptr) {

Node\* delNode = temp;

temp = temp->next;

delete delNode;

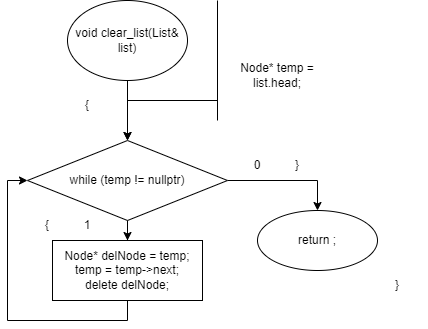
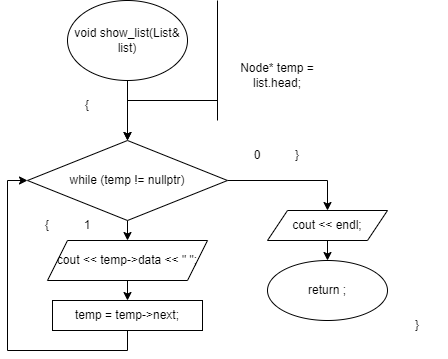
}

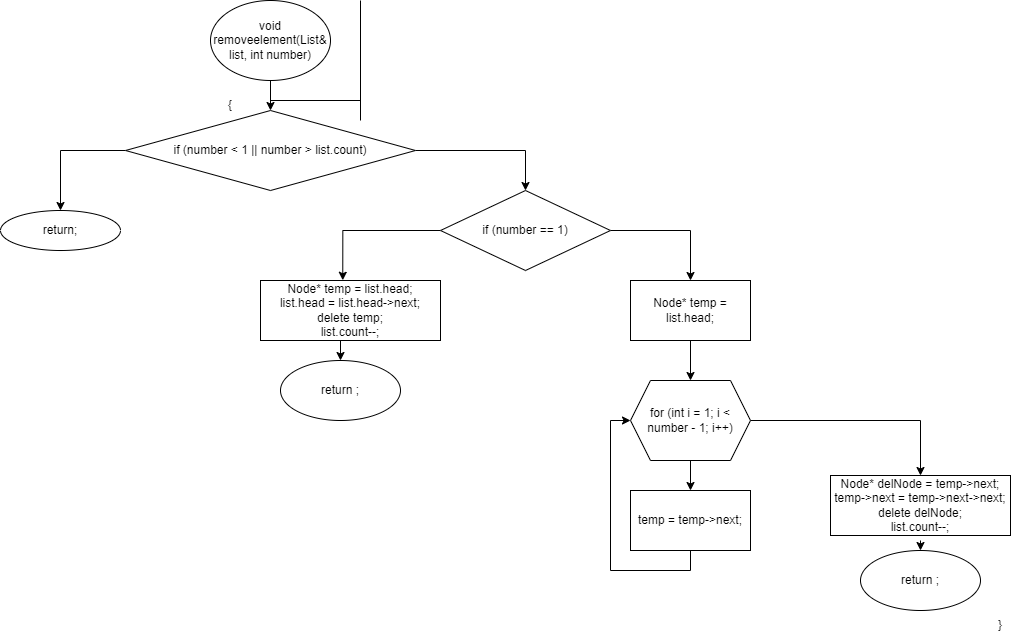
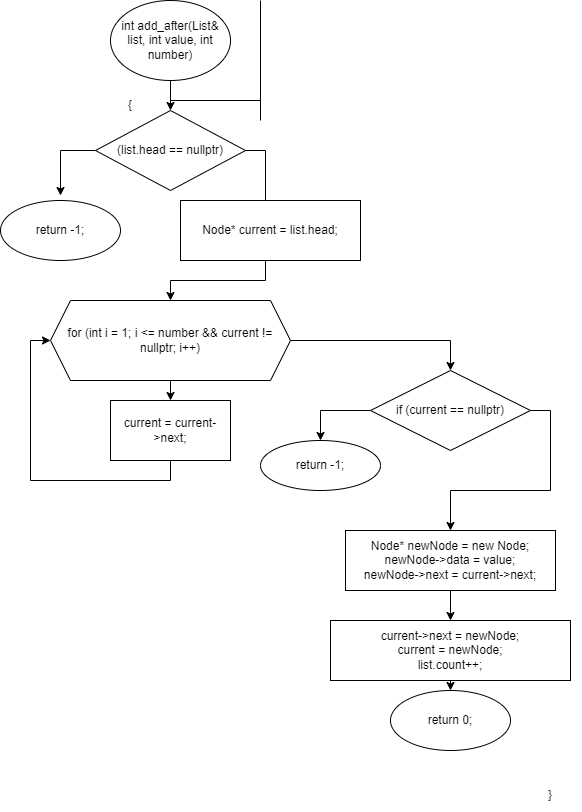
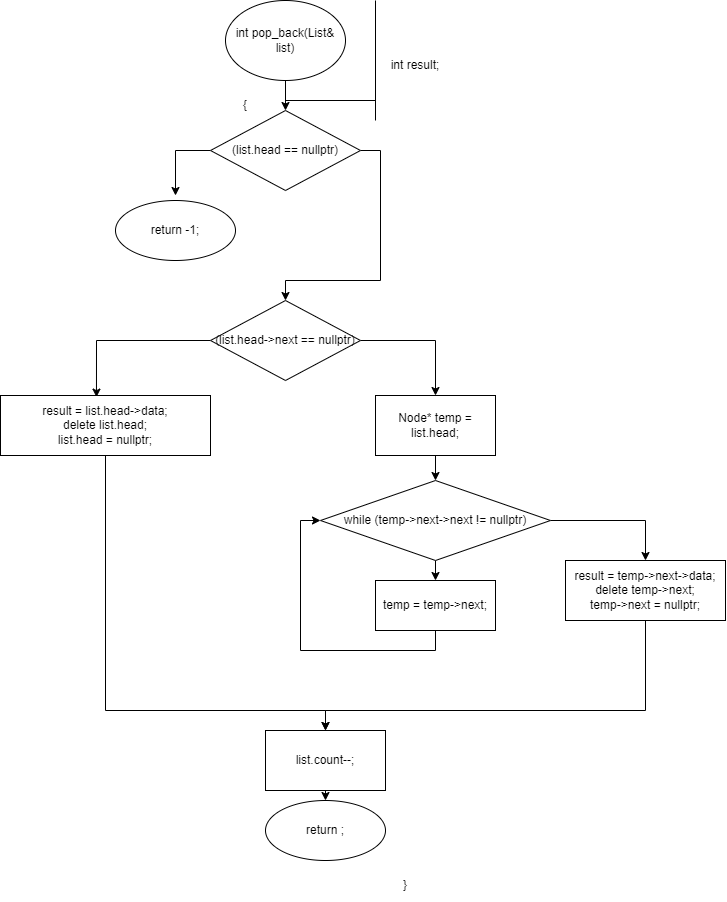
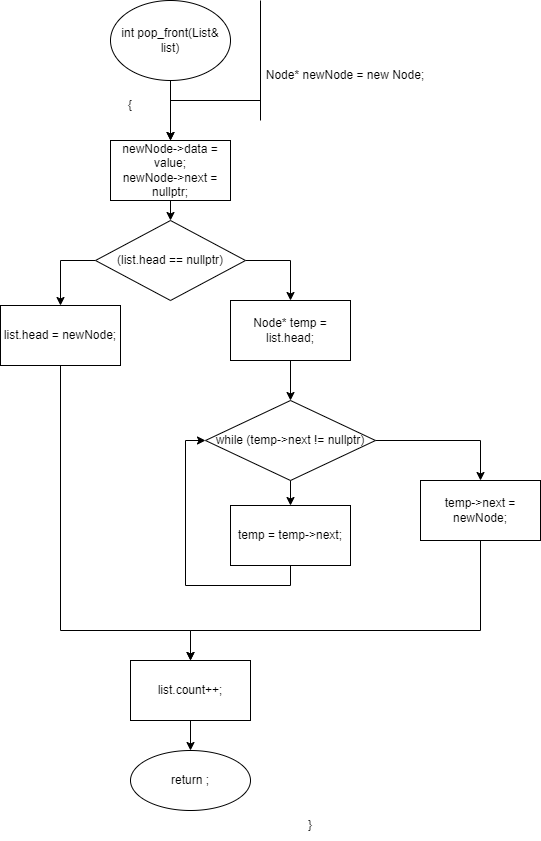
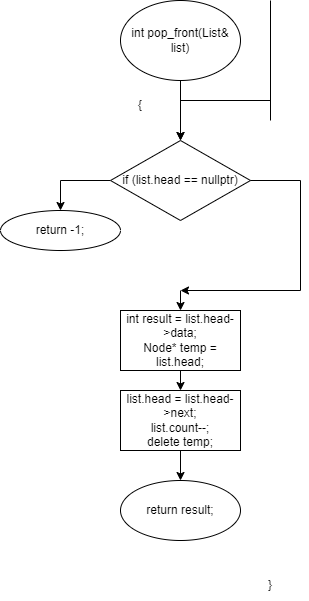
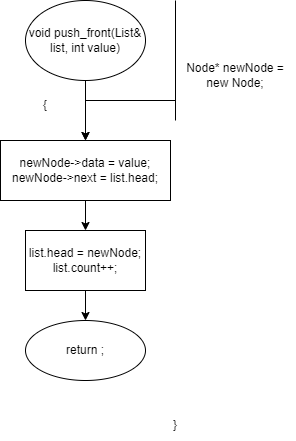
list.head = nullptr;

list.count = 0;

}

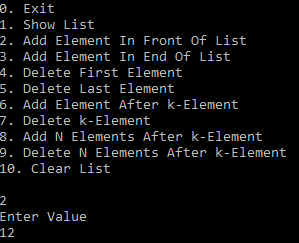
**Блок-схемы**

****

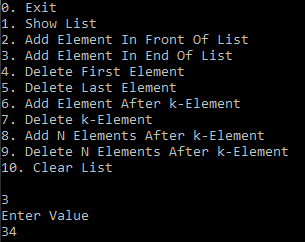
****

**Результат работы программы**

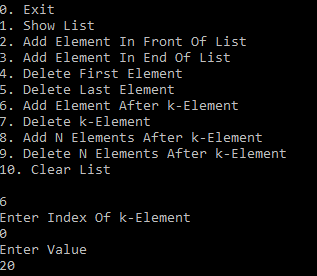
Добавление в начало



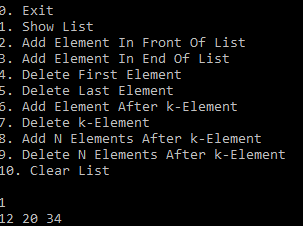
Добавление в конец



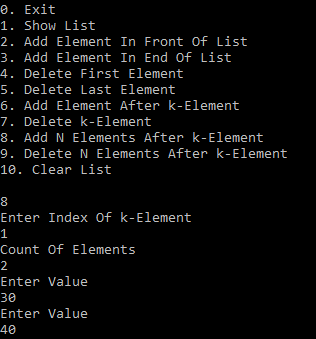
Добавление после k-элемента



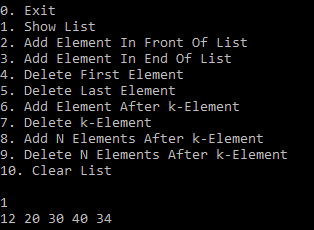
Показ списка



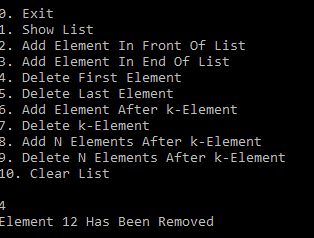
Добавление элементов после k-го



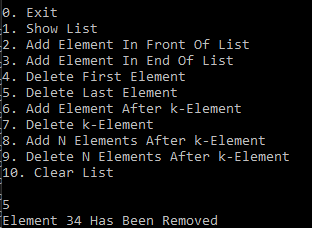
Показ списка



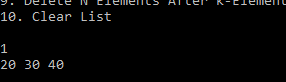
Удаление в начале списка



Удаление в конце списка



Показ списка



Удаление данных из списка

