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 ПО ЛАБОРАТОРНОЙ РАБОТЕ

Тема: "Списки, стеки, очереди"

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Мальчиков Дмитрий Григорьевич

Проверил

Доцент кафедры ИТАС

Полякова Ольга Андреевна

Г. Пермь-2023

Постановка задачи

А) Создать односвязный и двусвязный список и для каждого:  
1. Добавить элемент в начало списка  
2. Добавить элемент в конец списка  
3. Удалить элемент из начала списка  
4.Удалить элемент из конца списка  
5. Добавить К элементов перед элементом с заданным номером.  
6. Удалить К элементов перед элементом с заданным номером, начиная с заданного.

Б) Реализовать стеки и очереди через структуры.

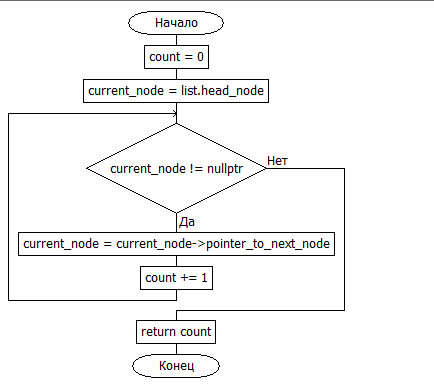
Односвязный список

Анализ задачи

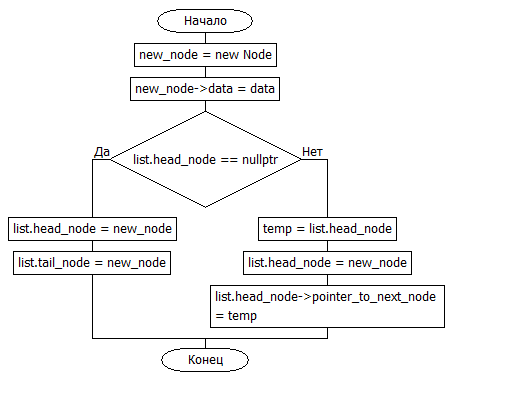
Чтобы добавить элемент в начало односвязного списка, нужно создать новый узел, указателем на следующий узел будет предыдущий головной, а теперешний головной будет этот новый узел.  
 Чтобы добавить элемент в конец списка, нужно указатель хвостового на следующий присвоить на новый узел, а новый узел теперь - хвостовой.  
 Чтобы удалить элемент из начала списка, нужно пеместить голову на следующий элемент, и теперешняя голова - это элемент после предыдущей головы.  
 Чтобы удалить элемент из конца списка нужно пройтись по всему списку до предпоследнего элемента. Указатель предпоследнего на следующий будет указывать не на хвост, а на nullptr (нулевой указатель), а теперешний хвостовой - этот предыдущий.  
 Чтобы добавить элемент в середину списка на K-тое место, нужно дойти до K-1 элемента, создать новый узел с данными, указатель нового узла указывает на предыдущий->следующий->следующий, указатель предыдущего указывает на новый узел.  
 Чтобы удалить элемент из середины списка на K-том месте, нужно дойти до K-1 элемента и теперь указатель этого узла на следующий - теперь этот->следующий->следующий (перескакиваем его).

Блок-схемы

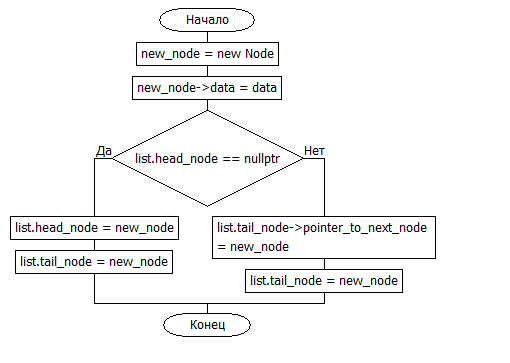
Int lenght()



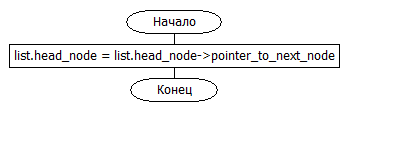
void pushFront()



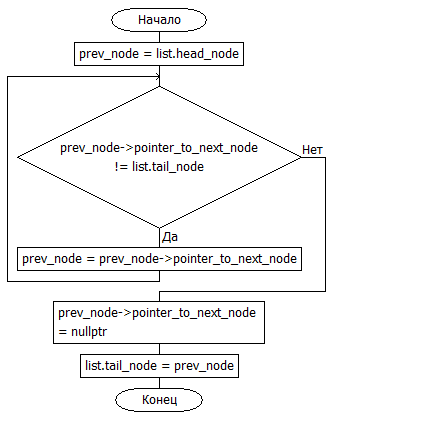
void pushBack()



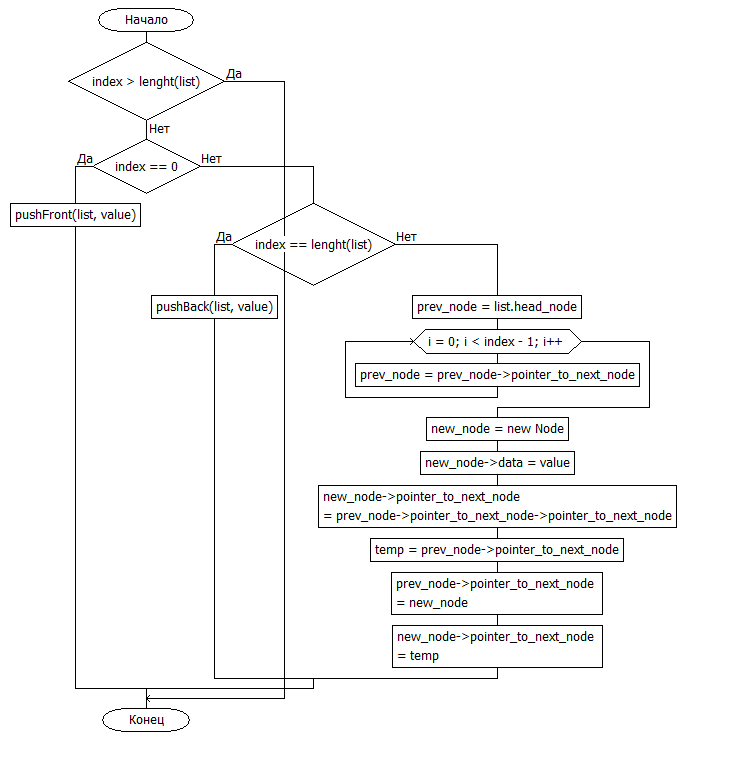
void popFront()



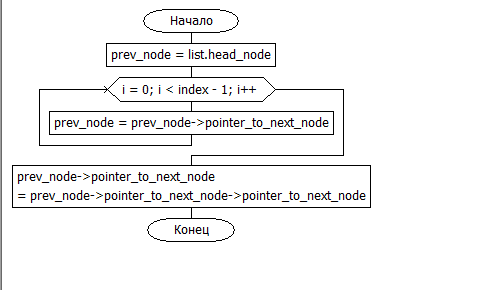
void popBack()



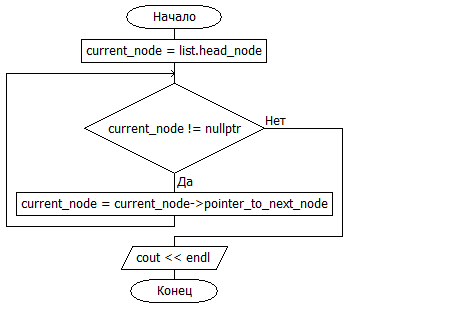
void adding()



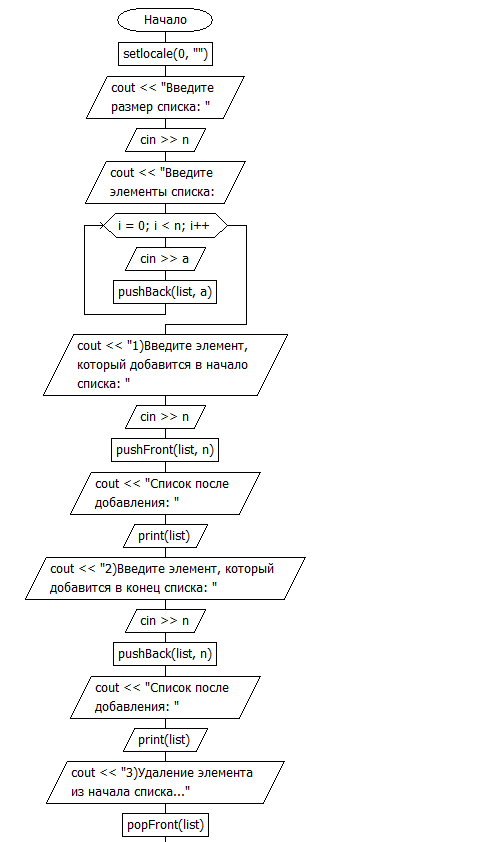
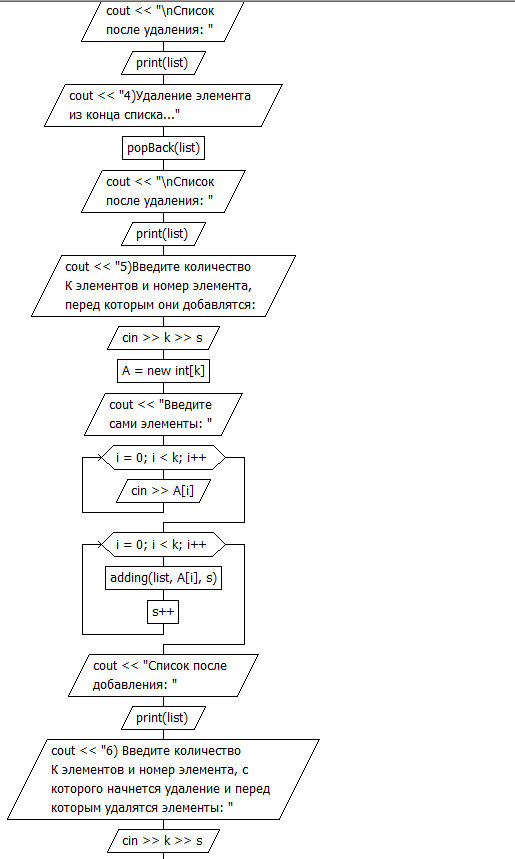
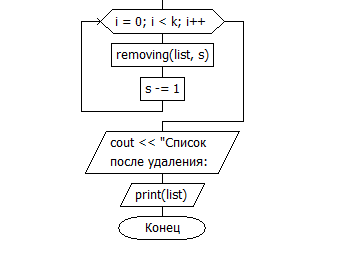
void removing()



void print()



Int main()

Программный код и результаты работы

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* pointer\_to\_next\_node = nullptr;

};

struct List {

Node\* head\_node = nullptr;

Node\* tail\_node = nullptr;

};

int lenght(List& list)

{

int count = 0;

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

while (current\_node != nullptr) {

current\_node = current\_node->pointer\_to\_next\_node;

count += 1;

}

return count;

}

void pushFront(List& list, const int& data) {

Node\* new\_node = new Node;

new\_node->data = data;

if (list.head\_node == nullptr) {

list.head\_node = new\_node;

list.tail\_node = new\_node;

}

else {

Node\* temp;

temp = list.head\_node;

list.head\_node = new\_node;

list.head\_node->pointer\_to\_next\_node = temp;

}

}

void pushBack(List& list, const int& data) {

Node\* new\_node = new Node;

new\_node->data = data;

if (list.head\_node == nullptr) {

list.head\_node = new\_node;

list.tail\_node = new\_node;

}

else {

list.tail\_node->pointer\_to\_next\_node = new\_node;

list.tail\_node = new\_node;

}

}

void popFront(List& list)

{

list.head\_node = list.head\_node->pointer\_to\_next\_node;

}

void popBack(List& list)

{

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

while (prev\_node->pointer\_to\_next\_node != list.tail\_node)

{

prev\_node = prev\_node->pointer\_to\_next\_node;

}

prev\_node->pointer\_to\_next\_node = nullptr;

list.tail\_node = prev\_node;

}

void adding(List& list, int value, int index)

{

if (index > lenght(list)) return;

if (index == 0)

{

pushFront(list, value);

}

else if (index == lenght(list))

{

pushBack(list, value);

}

else

{

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

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

{

prev\_node = prev\_node->pointer\_to\_next\_node;

}

Node\* new\_node = new Node;

new\_node->data = value;

new\_node->pointer\_to\_next\_node = prev\_node->pointer\_to\_next\_node->pointer\_to\_next\_node;

Node\* temp;

temp = prev\_node->pointer\_to\_next\_node;

prev\_node->pointer\_to\_next\_node = new\_node;

new\_node->pointer\_to\_next\_node = temp;

}

}

void removing(List& list, int index)

{

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

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

{

prev\_node = prev\_node->pointer\_to\_next\_node;

}

prev\_node->pointer\_to\_next\_node = prev\_node->pointer\_to\_next\_node->pointer\_to\_next\_node;

}

void print(List& list)

{

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

while (current\_node != nullptr) {

cout << current\_node->data << ' ';

current\_node = current\_node->pointer\_to\_next\_node;

}

cout << endl;

}

int main()

{

setlocale(0, "");

List list;

cout << "Введите размер списка: ";

int n;

cin >> n;

cout << "Введите элементы списка: ";

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

int a;

cin >> a;

pushBack(list, a);

}

cout << "1)Введите элемент, который добавится в начало списка: ";

cin >> n;

pushFront(list,n);

cout << "Список после добавления: ";

print(list);

cout << "2)Введите элемент, который добавится в конец списка: ";

cin >> n;

pushBack(list, n);

cout << "Список после добавления: ";

print(list);

cout << "3)Удаление элемента из начала списка...";

popFront(list);

cout << "\nСписок после удаления: ";

print(list);

cout << "4)Удаление элемента из конца списка...";

popBack(list);

cout << "\nСписок после удаления: ";

print(list);

cout << "5)Введите количество К элементов и номер элемента, перед которым они добавлятся: ";

int s, k;

cin >> k >> s;

int\* A = new int[k];

cout << "Введите сами элементы: ";

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

cin >> A[i];

}

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

adding(list, A[i], s);

s++;

}

cout << "Список после добавления: ";

print(list);

cout << "6) Введите количество К элементов и номер элемента, с которого начнется удаление и перед которым удалятся элементы: ";

cin >> k >> s;

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

removing(list, s);

s -= 1;

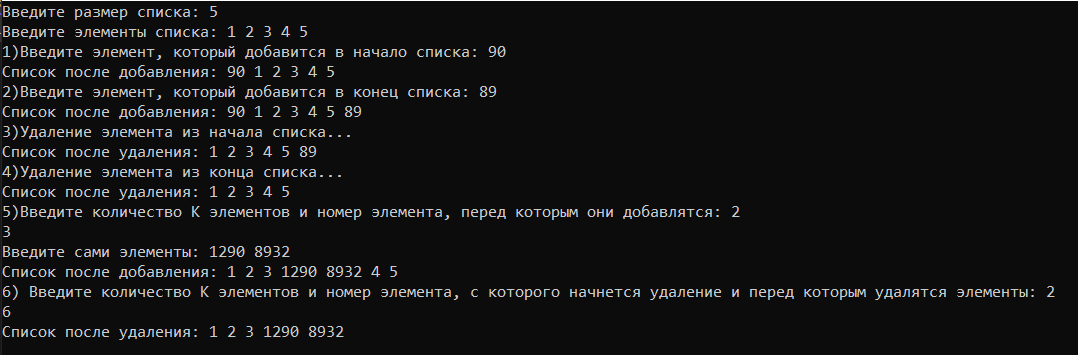
}

cout << "Список после удаления: ";

print(list);

}

Результаты работы

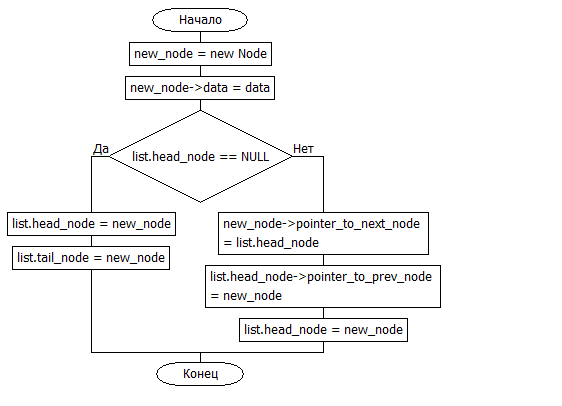


Двусвязные списки  
Анализ задачи

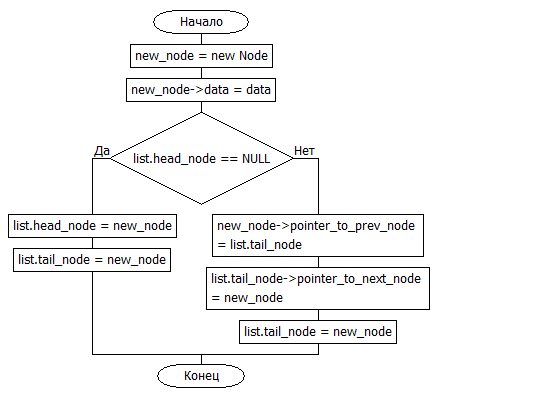
Чтобы добавить элемент в начало, создаём новый узел, указатель на следующий элемент этого узла указывает на текущий головной, указатель на предыдущий элемент головы указывает на новый, и делаем этот элемент головным   
 Чтобы добавить элемент в конец, создаём новый узел, указатель на предыдущий элемент этого узла указывает на хвостовой, указатель на следующий элемент хвостового указывает на новый, и делаем этот элемент хвостовым  
 Чтобы добавить элемент в середину на K-тое место, создаём новый узел, подходим либо к K-1 элемент слева либо к K+1 элементу справа и переприсваиваем указатели на предыдущие и следующие элементы.  
 Чтобы удалить элемент из начала, делаем головным элемент, на который указывала предыдущая голова, а информацию о прошлой голове удаляем  
 Чтобы удалить элемент из конца, нынешний хвост - предыдущий элемент за хвостом, удаляем информацию о предыдущем хвосте  
 Чтобы удалить элемент из середины на K-том месте, подходим либо к K-1 элементу слева либо к K+1 элементу справа, переприсваиваем указатели на предыдущие и следующие элементы и удаляем информацию об элементе K.

Блок-схемы

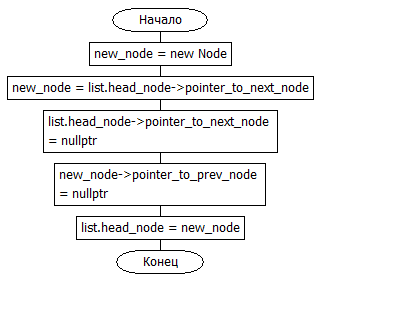
Void pushFront()



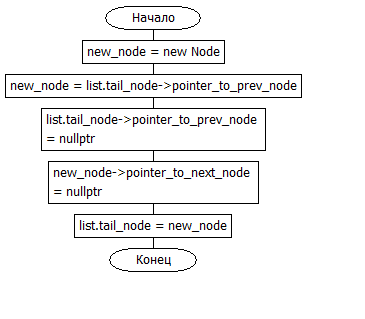
Void pushBack()



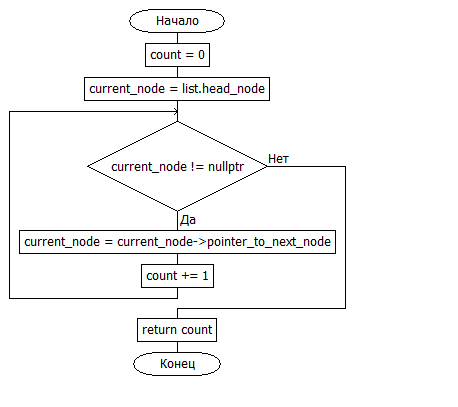
Void popFront()



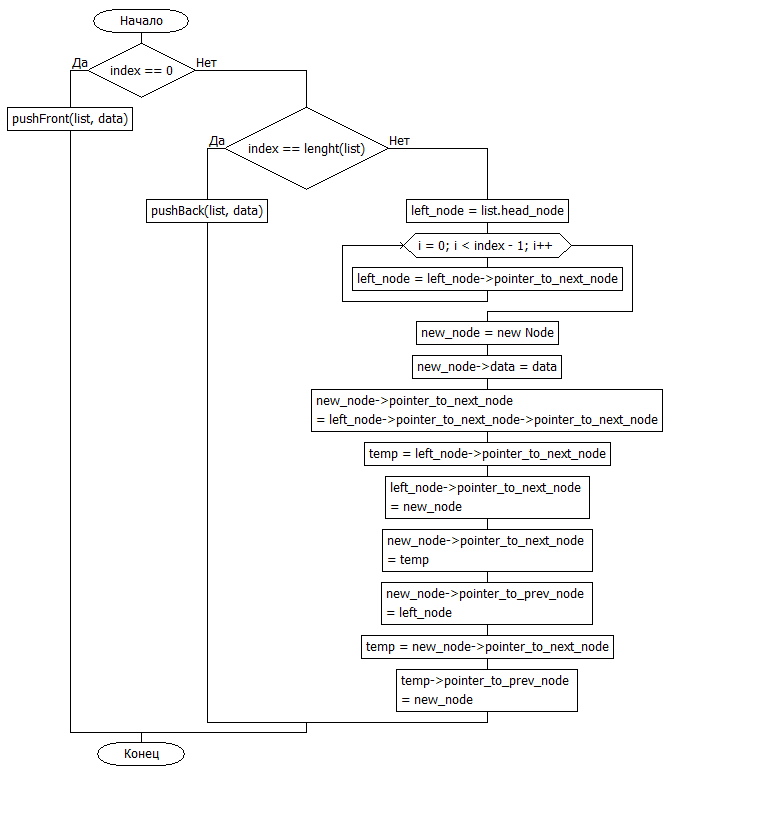
Void popBack()



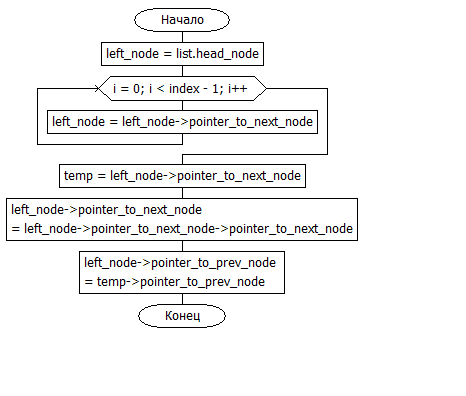
Int lenght()



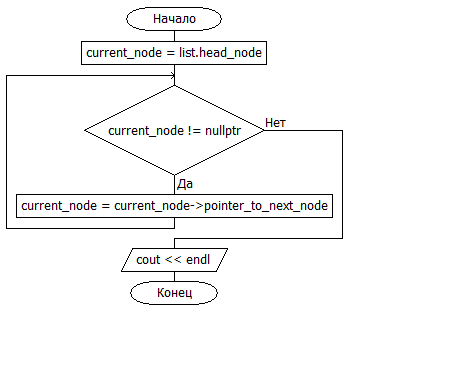
Void adding()



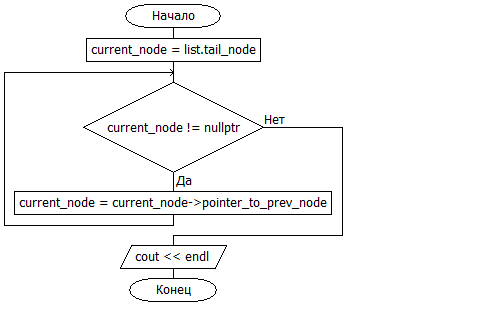
Void removing()



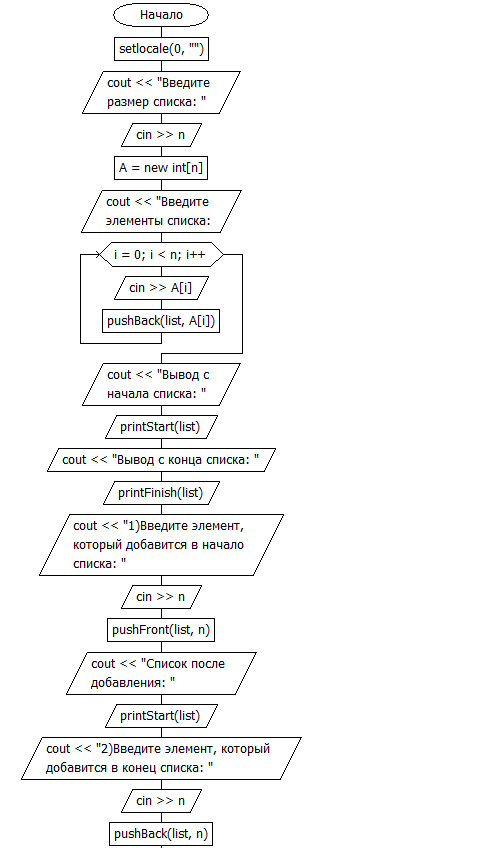
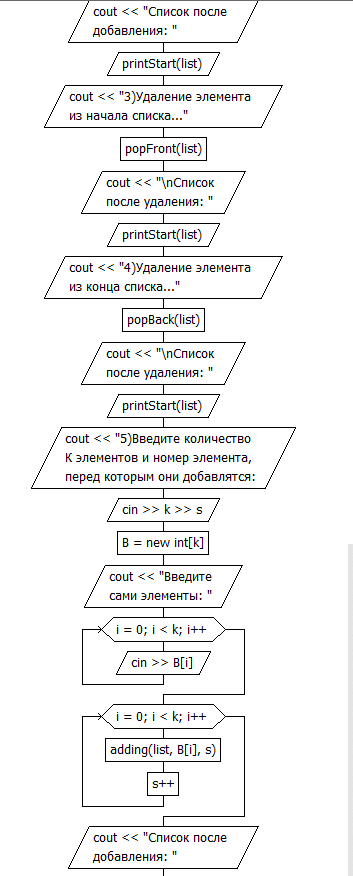
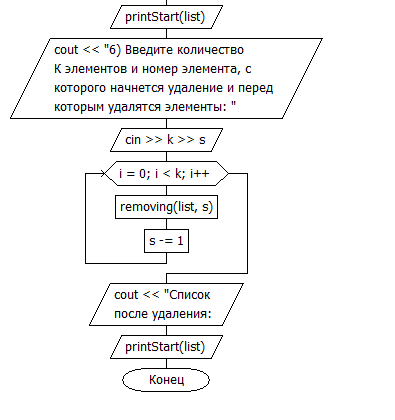
Void printStart()



Void printFinish()



Int main()

Программный код и результаты работы

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* pointer\_to\_prev\_node = nullptr;

Node\* pointer\_to\_next\_node = nullptr;

};

struct List {

Node\* head\_node = nullptr;

Node\* tail\_node = nullptr;

};

void pushFront(List& list, int data)

{

Node\* new\_node = new Node;

new\_node->data = data;

if (list.head\_node == NULL) {

list.head\_node = new\_node;

list.tail\_node = new\_node;

}

else {

new\_node->pointer\_to\_next\_node = list.head\_node;

list.head\_node->pointer\_to\_prev\_node = new\_node;

list.head\_node = new\_node;

}

}

void pushBack(List& list, int data)

{

Node\* new\_node = new Node;

new\_node->data = data;

if (list.head\_node == NULL) {

list.head\_node = new\_node;

list.tail\_node = new\_node;

}

else {

new\_node->pointer\_to\_prev\_node = list.tail\_node;

list.tail\_node->pointer\_to\_next\_node = new\_node;

list.tail\_node = new\_node;

}

}

void popFront(List& list)

{

Node\* new\_node = new Node;

new\_node = list.head\_node->pointer\_to\_next\_node;

list.head\_node->pointer\_to\_next\_node = nullptr;

new\_node->pointer\_to\_prev\_node = nullptr;

delete list.head\_node;

list.head\_node = new\_node;

}

void popBack(List& list)

{

Node\* new\_node = new Node;

new\_node = list.tail\_node->pointer\_to\_prev\_node;

list.tail\_node->pointer\_to\_prev\_node = nullptr;

new\_node->pointer\_to\_next\_node = nullptr;

delete list.tail\_node;

list.tail\_node = new\_node;

}

int lenght(List& list)

{

int count = 0;

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

while (current\_node != nullptr) {

current\_node = current\_node->pointer\_to\_next\_node;

count += 1;

}

return count;

}

void adding(List& list,int data, int index)

{

if (index == 0) pushFront(list, data);

else {

if (index == lenght(list)) pushBack(list, data);

else {

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

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

{

left\_node = left\_node->pointer\_to\_next\_node;

}

Node\* new\_node = new Node;

new\_node->data = data;

new\_node->pointer\_to\_next\_node = left\_node->pointer\_to\_next\_node->pointer\_to\_next\_node;

Node\* temp;

temp = left\_node->pointer\_to\_next\_node;

left\_node->pointer\_to\_next\_node = new\_node;

new\_node->pointer\_to\_next\_node = temp;

new\_node->pointer\_to\_prev\_node = left\_node;

temp = new\_node->pointer\_to\_next\_node;

temp->pointer\_to\_prev\_node = new\_node;

}

}

}

void removing(List& list, int index)

{

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

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

{

left\_node = left\_node->pointer\_to\_next\_node;

}

Node\* temp;

temp = left\_node->pointer\_to\_next\_node;

left\_node->pointer\_to\_next\_node = left\_node->pointer\_to\_next\_node->pointer\_to\_next\_node;

left\_node->pointer\_to\_prev\_node = temp->pointer\_to\_prev\_node;

}

void printStart(List& list)

{

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

while (current\_node != nullptr) {

cout << current\_node->data << ' ';

current\_node = current\_node->pointer\_to\_next\_node;

}

cout << endl;

}

void printFinish(List& list)

{

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

while (current\_node != nullptr) {

cout << current\_node->data << ' ';

current\_node = current\_node->pointer\_to\_prev\_node;

}

cout << endl;

}

int main()

{

setlocale(0, "");

int n, s, k;

List list;

cout << "Введите размер списка: "; cin >> n;

int\* A = new int[n];

cout << "Введите элементы списка: ";

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

cin >> A[i];

pushBack(list, A[i]);

}

cout << "Вывод с начала списка: ";

printStart(list);

cout << "Вывод с конца списка: ";

printFinish(list);

cout << "1)Введите элемент, который добавится в начало списка: ";

cin >> n;

pushFront(list, n);

cout << "Список после добавления: ";

printStart(list);

cout << "2)Введите элемент, который добавится в конец списка: ";

cin >> n;

pushBack(list, n);

cout << "Список после добавления: ";

printStart(list);

cout << "3)Удаление элемента из начала списка...";

popFront(list);

cout << "\nСписок после удаления: ";

printStart(list);

cout << "4)Удаление элемента из конца списка...";

popBack(list);

cout << "\nСписок после удаления: ";

printStart(list);

cout << "5)Введите количество К элементов и номер элемента, перед которым они добавлятся: ";

cin >> k >> s;

int\* B = new int[k];

cout << "Введите сами элементы: ";

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

cin >> B[i];

}

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

adding(list, B[i], s);

s++;

}

cout << "Список после добавления: ";

printStart(list);

cout << "6) Введите количество К элементов и номер элемента, с которого начнется удаление и перед которым удалятся элементы: ";

cin >> k >> s;

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

removing(list, s);

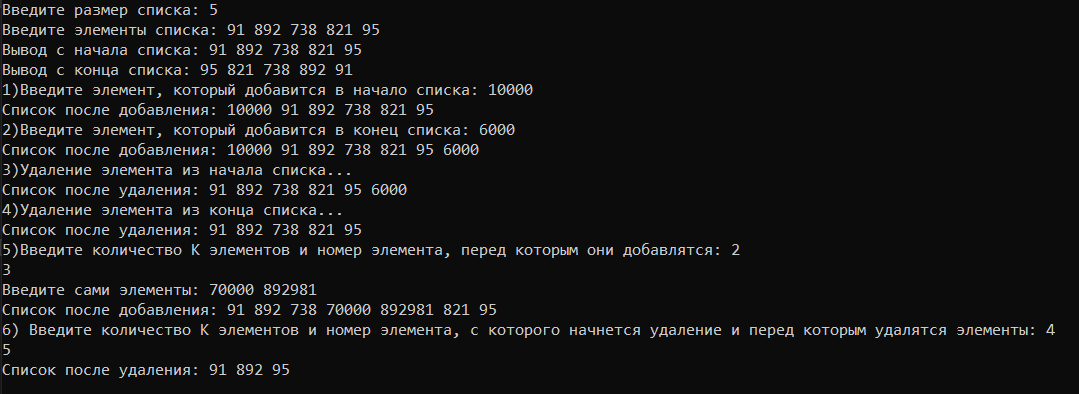
s -= 1;

}

cout << "Список после удаления: ";

printStart(list);

}

Результаты работы  


Стеки и очереди

Анализ задачи

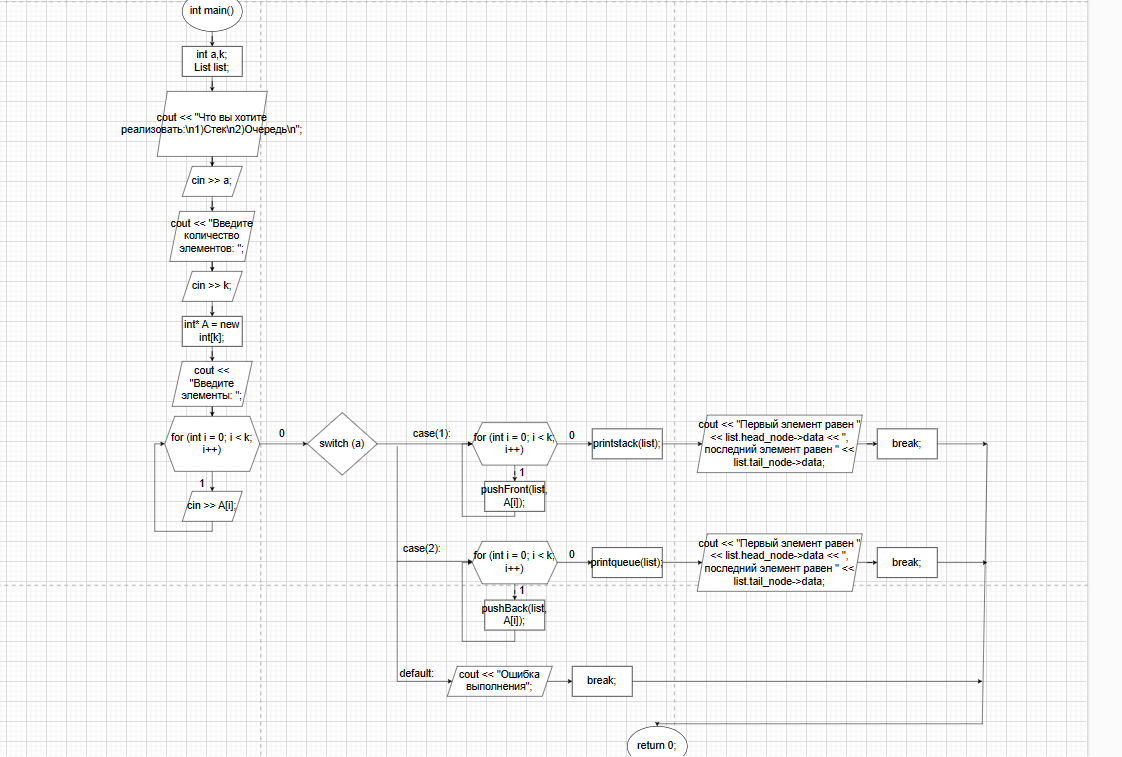
Работа со стеками и очередями предполагает работу с односвязными списками. Поэтому создаём структуру Node - для узлов и структуру List - для списков.

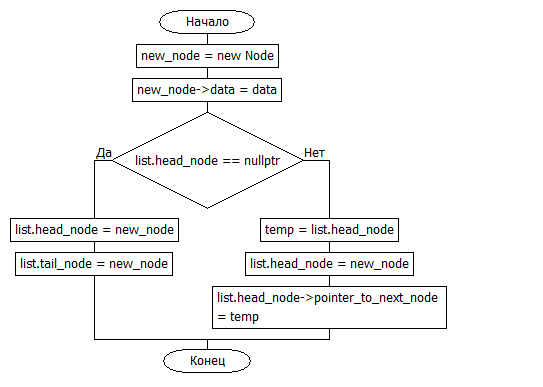
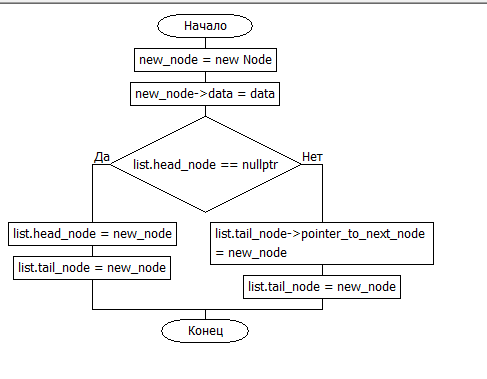
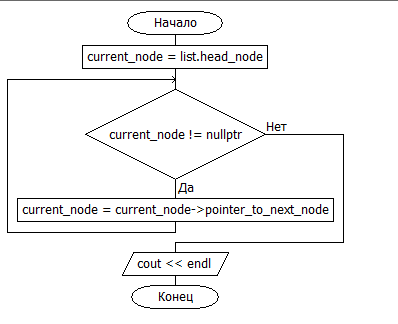
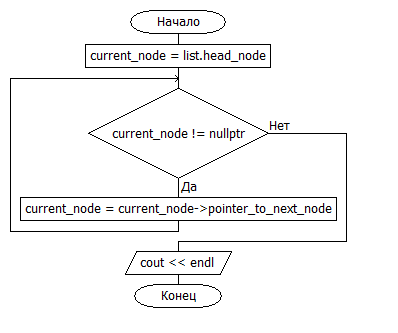
Для того чтобы добавлять элементы в стек, необходимо каждый новый элемент добавлять в начало списка (принцип “первый пришел - последний вышел”). Создадим для этого функцию pushFront().

Для очередей - каждый новый элемент добавляется в конец списка. Создадим для добавления элементов в конец списка функцию pushBack().

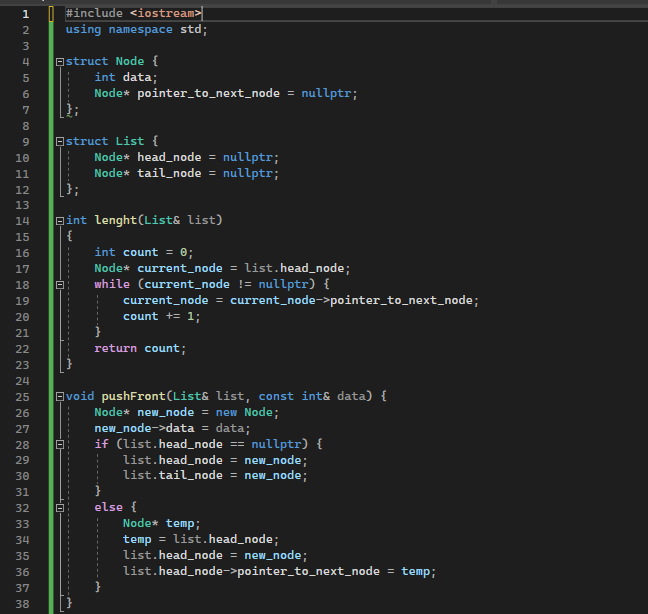
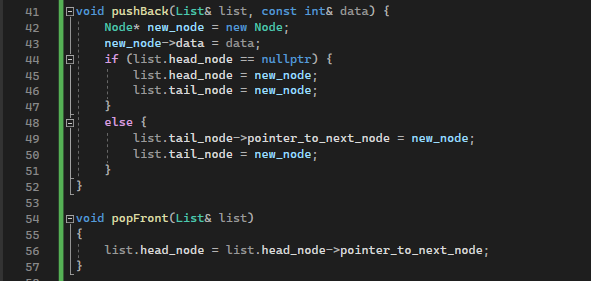
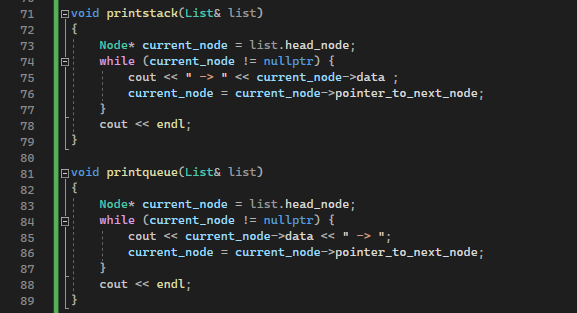
Для удобства записи также создадим функцию print(), который выводит на экран стек/очередь

Блок-схема



Программный код и результаты работы


Результаты работы

