Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

Лабораторная работа по теме “Динамические структуры ”

Выполнил: Студент группы РИС-23-3б

Асташин Дмитрий Алексеевич

Проверила: Доцент кафедры ИТАС

О. А. Полякова

2024

**Постановка задачи:**

Сформировать динамические структуры: односвязный и двусвязный списки, стек и очередь.

Удалить элемент с заданным ключом. Добавить по K элементов в начало и конец списка.

**Анализ задачи:**

Элемент каждой из динамических структур представляет собой узел, который имеет данные и указатели на следующий узел.

* Односвязный список

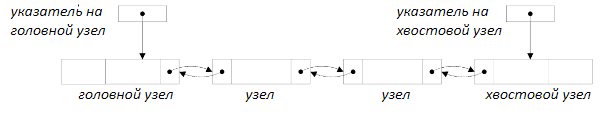


Первый узел – головной.

При добавлении элемента указатель предыдущего указываем на новый элемент, а указатель нового указываем на следующий.

При удалении указатель предыдущего указывается на следующий элемент после удаляемого.

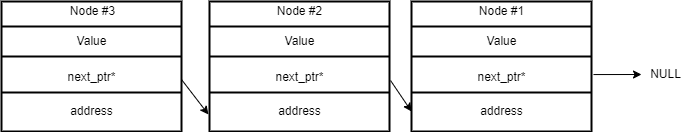
* Двусвязный список

****

Первый узел головной, последний хвостовой.

В отличие от односвязного списка, по двусвязному можно идти в обратном направлении, так как узел имеет два указателя: на прошлый и следующий.

* Стек



**Работает по принципу: последний пришел – первый ушел**

Стек представляет собой стакан, поэтому чтобы добавить или удалить элемент, который находится в середине, необходимо достать все элементы выше него (например, переместить их в другой стек), затем добавить или удалить элемент и вернуть данные, которые достали.

* Очередь

**Работает по принципу: первый пришел – первый ушел**

Чтобы удалить элемент, необходимо переместить его в самое начало очереди.

Новый элемент добавляется в конец очереди, но если нужно добавить элемент между двумя существующими, то передвигаем первый из этих двух элементов в конец и добавляем новый.



**Односвязный список:**

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

struct Node {

string data;

Node\* next;

};

struct List {

Node\* head = nullptr;

};

// Функция добавления в конец

void pushBack(List& list, string data) {

Node\* new\_node = new Node;

new\_node->data = data;

new\_node->next = nullptr;

if (list.head == nullptr) {

list.head = new\_node;

}

else {

Node\* current\_node = list.head;

while (current\_node->next != nullptr) {

current\_node = current\_node->next;

}

current\_node->next = new\_node;

}

}

// Функция добавления в начало

void pushFront(List& list, string data) {

Node\* new\_node = new Node;

new\_node->data = data;

new\_node->next = list.head;

list.head = new\_node;

}

// Функция удаления всех узлов с заданным значением

void del(List& list, string data) {

while (list.head != nullptr && list.head->data == data) {

Node\* temp = list.head;

list.head = list.head->next;

delete temp;

}

Node\* curr = list.head;

while (curr != nullptr && curr->next != nullptr) {

if (curr->next->data == data) {

Node\* temp = curr->next;

curr->next = curr->next->next;

delete temp;

}

else {

curr = curr->next;

}

}

}

// Функция печати списка

void print(List& list) {

Node\* current\_node = list.head;

if (current\_node == nullptr) {

cout << "No data!" << endl;

}

while (current\_node != nullptr) {

cout << " -| " << current\_node->data << " |- ";

current\_node = current\_node->next;

}

}

// Функция удаления списка

void clear(List& list) {

Node\* del\_node = list.head;

while (del\_node != nullptr) {

Node\* next\_node = del\_node->next;

delete del\_node;

del\_node = next\_node;

}

list.head = nullptr;

}

// Функция записи структуры в файл

void addToFile(const char\* filename, List& list) {

// Открытие файла для записи

ofstream File(filename);

// Запись в файл данных каждого узла

Node\* current\_node = list.head;

while (current\_node != nullptr) {

File << current\_node->data << endl;

current\_node = current\_node->next;

}

File.close();

}

// Функция восстановления структуры из файла

void recover(const char\* filename, List& list) {

// Очищаем текущий список перед восстановлением

clear(list);

// Открытие файла для чтения

ifstream File(filename);

string data;

while (getline(File, data)) {

pushBack(list, data);

}

File.close();

}

int main() {

List list;

int k;

cout << "Elements num(k): ";

cin >> k;

cout << "---------------------------------" << endl;

cin.ignore();

string line;

// Добавление в конец

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

cout << "Push Back element: ";

getline(cin, line);

pushBack(list, line);

}

cout << "---------------------------------" << endl;

// Добавление в начало

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

cout << "Push Forward element: ";

getline(cin, line);

pushFront(list, line);

}

cout << "---------------------------------" << endl;

// Вывод заполненного списка

cout << "List: ";

print(list);

cout << endl;

cout << "---------------------------------" << endl;

// Удаление элемента

cout << "Delete element: ";

getline(cin, line);

del(list, line);

cout << "---------------------------------" << endl;

// Вывод обработанного списка

cout << "List: ";

print(list);

cout << endl;

cout << "---------------------------------" << endl;

//Запись в файл

cout << "Add data to file!" << endl;

addToFile("file.dat", list);

cout << "---------------------------------" << endl;

// Очищаем память

clear(list);

cout << "Cleared!" << endl;

cout << "---------------------------------" << endl;

// Восстановление данных

cout << "Restoring data from a file!" << endl;

recover("file.dat", list);

cout << "---------------------------------" << endl;

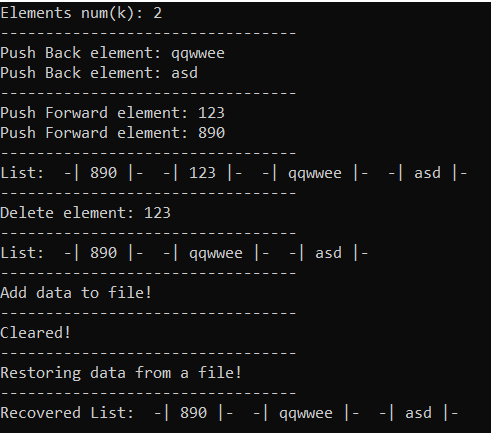
cout << "Recovered List: ";

print(list);

cout << endl;

return 0;

}

****

**Двусвязный список**

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

struct Node {

string data;

Node\* next;

Node\* prev;

};

struct List {

Node\* head = nullptr;

Node\* tail = nullptr;

};

// Функция добавления в конец

void pushBack(List& list, string data) {

Node\* new\_node = new Node;

new\_node->data = data;

new\_node->next = nullptr;

new\_node->prev = nullptr;

if (list.head == nullptr) {

list.head = new\_node;

list.tail = new\_node;

}

else {

list.tail->next = new\_node;

new\_node->prev = list.tail;

list.tail = new\_node;

}

}

// Функция добавления в начало

void pushFront(List& list, string data) {

Node\* new\_node = new Node;

new\_node->data = data;

new\_node->next = list.head;

new\_node->prev = nullptr;

if (list.head != nullptr) {

list.head->prev = new\_node;

}

list.head = new\_node;

if (list.tail == nullptr) {

list.tail = new\_node;

}

}

// Функция удаления всех узлов с заданным значением

void del(List& list, string data) {

Node\* current\_node = list.head;

while (current\_node != nullptr) {

if (current\_node->data == data) {

if (current\_node->prev != nullptr) {

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

}

if (current\_node->next != nullptr) {

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

}

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

list.head = current\_node->next;

}

if (current\_node == list.tail) {

list.tail = current\_node->prev;

}

Node\* temp = current\_node;

current\_node = current\_node->next;

delete temp;

}

else {

current\_node = current\_node->next;

}

}

}

// Функция печати списка

void print(List& list) {

Node\* current\_node = list.head;

if (current\_node == nullptr) {

cout << "No data!" << endl;

}

while (current\_node != nullptr) {

cout << " -| " << current\_node->data << " |- ";

current\_node = current\_node->next;

}

}

// Функция печати списка с конца до начала

void printReverse(List& list) {

Node\* current\_node = list.tail;

if (current\_node == nullptr) {

cout << "No data!" << endl;

}

while (current\_node != nullptr) {

cout << " -| " << current\_node->data << " |- ";

current\_node = current\_node->prev;

}

}

// Функция удаления списка

void clear(List& list) {

while (list.head != nullptr) {

Node\* temp = list.head;

list.head = list.head->next;

delete temp;

}

}

// Функция записи структуры в файл

void addToFile(const char\* filename, List& list) {

// Открытие файла для записи

ofstream File(filename);

// Запись в файл данных каждого узла

Node\* current\_node = list.head;

while (current\_node != nullptr) {

File << current\_node->data << endl;

current\_node = current\_node->next;

}

File.close();

}

// Функция восстановления структуры из файла

void recover(const char\* filename, List& list) {

// Очищаем текущий список перед восстановлением

clear(list);

// Открытие файла для чтения

ifstream File(filename);

string data;

while (getline(File, data)) {

pushBack(list, data);

}

File.close();

}

int main() {

List list;

int k;

cout << "Elements num(k): ";

cin >> k;

cout << "---------------" << endl;

cin.ignore();

string line;

// Добавление в конец

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

cout << "Push Back element: ";

getline(cin, line);

pushBack(list, line);

}

cout << "---------------" << endl;

// Добавление в начало

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

cout << "Push Forward element: ";

getline(cin, line);

pushFront(list, line);

}

cout << "---------------------------------" << endl;

// Вывод заполненного списка

cout << "List: ";

print(list);

cout << endl;

cout << "Reverce list: ";

printReverse(list);

cout << endl;

cout << "---------------------------------" << endl;

// Удаление элемента

cout << "Delete element: ";

getline(cin, line);

del(list, line);

cout << "---------------------------------" << endl;

// Вывод обработанного списка

cout << "List: ";

print(list);

cout << endl;

cout << "---------------------------------" << endl;

//Запись в файл

cout << "Add data to file!" << endl;

addToFile("file.dat", list);

cout << "---------------------------------" << endl;

// Очищаем память

clear(list);

cout << "Cleared!" << endl;

cout << "---------------------------------" << endl;

// Восстановление данных

cout << "Restoring data from a file!" << endl;

recover("file.dat", list);

cout << "---------------------------------" << endl;

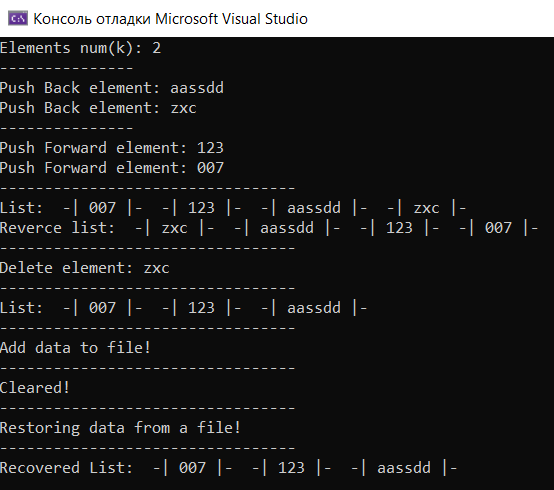
cout << "Recovered List: ";

print(list);

cout << endl;

return 0;

}

****

**Стек**

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

template <typename T>

struct Node

{

T data;

Node<T>\* next\_ptr;

};

template <typename T>

struct Stack

{

Node<T>\* head;

int size;

};

// Функция, которая устанавливает поля по умолчанию узлу

template <typename T>

void SetNode(Node<T>\*& node, T data, Node<T>\* next\_ptr = nullptr)

{

node = new Node<T>();

node->next\_ptr = next\_ptr;

node->data = data;

}

// Функция, которая инициализирует поля стека базовыми значениями

template <typename T>

void SetStack(Stack<T>& tmp)

{

tmp.head = nullptr;

tmp.size = 0;

}

// Функция добавления

template <typename T>

void Push(Stack <T>& tmp, const T& data)

{

// Создается новый узел

Node<T>\* new\_node;

SetNode(new\_node, data, tmp.head);

tmp.head = new\_node;

tmp.size++;

}

// Функция удаления

template <typename T>

void pop(Stack <T>& stack, const T& data)

{

if (stack.head == nullptr)

return;

Node<T>\* temp = stack.head;

stack.head = temp->next\_ptr;

delete temp;

stack.size--;

}

// Функция печати

template <typename T>

void print(Stack<T>& stack) {

if (stack.head == nullptr) {

cout << "Stack is empty" << endl;

return;

}

Node<T>\* current\_node = stack.head;

while (current\_node != nullptr) {

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

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

}

cout << endl;

}

// Функция очистки стека

template <typename T>

void clear(Stack<T>& stack) {

Node<T>\* current\_node = stack.head;

while (current\_node != nullptr) {

Node<T>\* temp = current\_node;

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

delete temp;

}

stack.head = nullptr;

stack.size = 0;

}

// Функция для перемещения элементов в дополнительный Стек

template <typename T>

void moveAllToHelpStack(Stack<T>& stack, Stack<T>& helpStack) {

Node<T>\* current\_node = stack.head;

while (current\_node != nullptr) {

Push(helpStack, current\_node->data); // Исправлено

current\_node = current\_node->next\_ptr; // Сдвиг на следующий узел

}

clear(stack);

stack.size = 0; // Уменьшение размера main\_stack

cout << "-----------------------------------------" << endl;

cout << "Data Move" << endl;

cout << "-----------------------------------------" << endl;

}

template <typename T>

void moveSeveral(Stack<T>& stack, Stack<T>& helpStack, string data)

{

Node<T>\* current\_node = stack.head;

while (current\_node->data != data) {

Node<T>\* temp = current\_node;

Push(helpStack, current\_node->data);

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

stack.head = stack.head->next\_ptr;

delete temp;

stack.size--;

}

cout << "-----------------------------------------" << endl;

cout << "Data Move" << endl;

cout << "-----------------------------------------" << endl;

}

int main() {

// Инициализация

Stack<string> main\_stack;

Stack<string> helpStack;

SetStack(main\_stack);

SetStack(helpStack);

int numberStack, k;

string data;

// Количество начальных элементов

cout << "Number of elements: ";

cin >> numberStack;

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

cout << "Input element: ";

cin >> data;

Push(main\_stack, data);

}

cout << "Stack: ";

print(main\_stack);

cout << endl;

moveAllToHelpStack(main\_stack, helpStack);

print(main\_stack);

cout << endl;

cout << "Number k = ";

cin >> k;

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

cout << "Data: ";

cin >> data;

Push(main\_stack, data);

}

moveAllToHelpStack(helpStack, main\_stack);

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

cout << "Data: ";

cin >> data;

Push(main\_stack, data);

}

print(main\_stack);

cout << endl;

cout << "-----------------------------------------" << endl;

cout << "Delete element: ";

cin >> data;

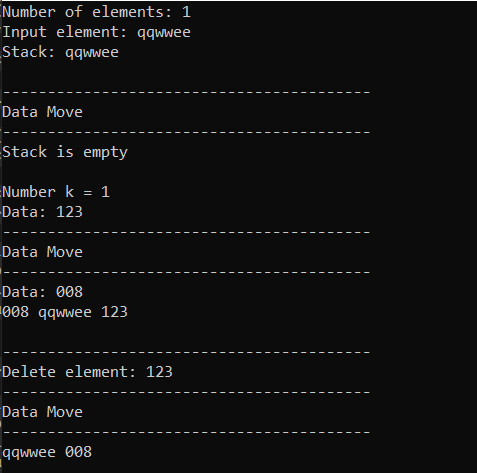
moveSeveral(main\_stack, helpStack, data);

print(helpStack);

clear(main\_stack);

clear(helpStack);

}

****

**Очередь**

#include <iostream>

#include <string>

using namespace std;

template <typename T>

struct Node

{

T data;

Node<T>\* next;

};

template <typename T>

struct Queue

{

int size;

Node<T>\* head;

Node<T>\* tail;

};

template <typename T>

void init\_queue(Queue<T>& Q, const T& data)

{

Node<T>\* new\_node = new Node<T>();

new\_node->data = data;

Q.head = new\_node;

Q.tail = new\_node;

Q.size = 1;

}

template <typename T>

void pushBack(Queue<T>& Q, const T& data)

{

Node<T>\* new\_node = new Node<T>();

Q.size++;

new\_node->data = data;

new\_node->next = nullptr;

Q.tail->next = new\_node;

Q.tail = new\_node;

}

template <typename T>

void new\_queue(Queue<T>& Q, int n)

{

T data;

cout << "Input elemnt: ";

cin >> data;

init\_queue(Q, data);

for (int i = 2; i <= n; i++)

{

cout << "Input element: ";

cin >> data;

pushBack(Q, data);

}

}

template <typename T>

void Pop(Queue<T>& Q)

{

Node<T>\* temp = Q.head;

Q.head = temp->next;

Q.size--;

delete temp;

}

template <typename T>

void deleteElement(Queue<T>& Q, const T& key)

{

for (int i = 1; i < Q.size; i++) {

if (Q.head->data != key) {

pushBack(Q, Q.head->data);

Pop(Q);

}

else {

Pop(Q);

}

}

}

template <typename T>

void pushFront(Queue <T>& Q, const T& key)

{

pushBack(Q, key);

print(Q);

for (int i = 1; i < Q.size; i++) {

pushBack(Q, Q.head->data);

Pop(Q);

print(Q);

}

}

template <typename T>

void print(Queue <T>& Q)

{

Node<T>\* current\_node = Q.head;

cout << "Start -> ";

while (current\_node != nullptr) {

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

current\_node = current\_node->next;

}

cout << "Final" << endl;

}

template <typename T>

void clear(Queue<T>& Q) {

Node<T>\* current\_node = Q.head;

while (current\_node != nullptr) {

Node<T>\* temp = current\_node;

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

delete temp;

}

Q.head = nullptr;

Q.size = 0;

}

int main() {

Queue<string> MyQ;

int size;

int k;

string data;

cout << "Input size: ";

cout << endl;

cin >> size;

new\_queue(MyQ, size);

print(MyQ);

cout << "Input k: ";

cin >> k;

cout << endl;

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

cout << "Input elemnt: ";

cin >> data;

pushFront(MyQ, data);

}

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

cout << "Input elemnt: ";

cin >> data;

pushBack(MyQ, data);

print(MyQ);

}

cout << "Input element for delete: ";

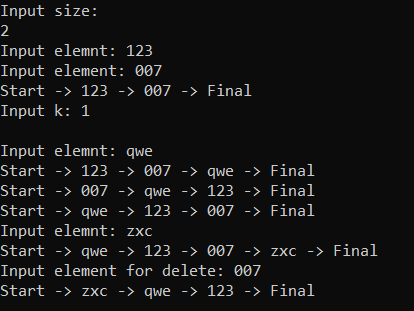
cin >> data;

deleteElement(MyQ, data);

print(MyQ);

clear(MyQ);

}

****

**https://github.com/DimetriusAsti/Dynamic/tree/main**