МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации

**

**ОТЧЁТ**

**по практической работе №1**

**«**Разработка СУБД**»**

**по дисциплине: «***Программирование***»**

Выполнил:Проверил:

Студент гр. «АБс-324», «АВТФ» *доцент кафедры ЗИ*

*Клаус В. А. Архипова А. Б.*

«\_\_» \_\_\_\_\_\_ 2024 г.«\_\_\_» \_\_\_\_\_\_ 2024 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

Новосибирск 2024

**Задание:** Требуется реализовать реляционную СУБД с поддержкой запросов на языке SQL. СУБД использует единственный тип данных - строка.

Требуется поддержать следующие SQL выражения:

SELECT <> FROM <> - выборка

WHERE и операторы OR , AND - фильтрация

INSERT INTO - вставка данных в таблицы

DELETE FROM - удаление данных из таблицы

# Модули программы

## schema.json:

{

"names": "БдАвтосалона",

"tuples\_limit": 5,

"structure": {

"cars": ["марка", "год", "пробег", "цена"],

"users": ["фамилия", "имя", "марка"]

}

}

## headers:

include.h

#pragma once

using namespace std;

#include <iostream>

#include <fstream>

#include <sstream>

list.h

#pragma once

#include "includes.h"

template<typename T>

struct Node {

T data;

Node\* next;

Node(T value); // конструктор узла

};

template<typename T>

struct SinglyLinkedList {

Node<T>\* head;

int size = 0;

SinglyLinkedList(); // конструктор листа

~SinglyLinkedList(); // деструктор

void print(); // вывод списка

void push\_front(T value); // добавление элемента в начало

void push\_back(T value); // добавление элемента в конец

void pop\_front(); // удаление элемента с начала

void pop\_back(); // удаление элемента с конца

void remove(T value); // удаление элемента по значению

void replace(int index, T newValue); // ф-ия замены элемента по индексу

int getindex(T value); // поиск элемента по значению

T getvalue(int index); // поиск элемента по индексу

};

template <typename T>

struct Node2 {

T data;

Node2\* next;

Node2\* prev;

Node2(T value); // конструктор узла

};

template <typename T>

struct DoublyLinkedList {

Node2<T>\* head;

Node2<T>\* tail;

int size = 0;

DoublyLinkedList(); // конструктор листа

~DoublyLinkedList(); // деструктор

void print(); // вывод списка

void push\_front(T value); // добавление элемента в начало

void push\_back(T value); // добавление элемента в конец

void pop\_front(); // удаление элемента в начале

void pop\_back(); // удаление элемента в конце

void remove(T value); // удаление элемента по значению

bool find(T value); // поиска элемента по значению

};

#include "../src/list.cpp" // Включаем реализацию шаблона

## 

## 

## файлы cpp:

list.cpp

#include "../include/list.h"

template<typename T>

Node<T>::Node(T value) : data(value), next(nullptr) {}

template<typename T>

SinglyLinkedList<T>::SinglyLinkedList() : head(nullptr) {}

template<typename T>

void SinglyLinkedList<T>::print() {

Node<T>\* current = head;

while (current) {

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

current = current->next;

}

cout << endl;

}

template<typename T>

void SinglyLinkedList<T>::push\_front(T value) {

Node<T>\* newNode = new Node<T>(value);

newNode->next = head;

head = newNode;

size++;

}

template<typename T>

void SinglyLinkedList<T>::push\_back(T value) {

Node<T>\* newNode = new Node<T>(value);

if (head == nullptr) {

head = newNode;

} else {

Node<T>\* current = head;

while (current->next) {

current = current->next;

}

current->next = newNode;

}

size++;

}

template<typename T>

void SinglyLinkedList<T>::pop\_front() {

if (head == nullptr) return;

Node<T>\* temp = head;

head = head->next;

delete temp;

size--;

}

template<typename T>

void SinglyLinkedList<T>::pop\_back() {

if (head == nullptr) return;

if (!head->next) { // если только 1 элемент

delete head;

head = nullptr;

return;

}

Node<T>\* current = head;

while (current->next && current->next->next) {

current = current->next;

}

delete current->next; // Удаляем последний элемент

current->next = nullptr; // Обнуляем указатель

size--;

}

template<typename T>

void SinglyLinkedList<T>::remove(T value) {

if (head == nullptr) return;

if (head->data == value) {

pop\_front();

return;

}

Node<T>\* current = head;

while (current->next) {

if (current->next->data == value) {

Node<T>\* temp = current->next;

current->next = current->next->next;

delete temp;

return;

}

current = current->next;

}

size--;

}

template<typename T>

void SinglyLinkedList<T>::replace(int index, T newValue) {

if (index < 0 || index >= size) {

cout << "Index out of bounds." << endl;

return;

}

Node<T>\* current = head;

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

current = current->next;

}

current->data = newValue;

}

template<typename T>

int SinglyLinkedList<T>::getindex(T value) {

Node<T>\* current = head;

int index = 0;

while (current) {

if (current->data == value) {

return index; // Элемент найден, возвращаем индекс

}

current = current->next;

index++;

}

return -1; // Если элемент не найден, возвращаем -1

}

template<typename T>

T SinglyLinkedList<T>::getvalue(int index) {

if (index < 0 || index >= size) {

throw out\_of\_range("Index out of range");

}

Node<T>\* current = head;

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

current = current->next;

}

return current->data;

}

template<typename T>

SinglyLinkedList<T>::~SinglyLinkedList() {

while (head) {

pop\_front();

}

}

template<typename T>

Node2<T>::Node2(T value) : data(value), next(nullptr), prev(nullptr) {}

template<typename T>

DoublyLinkedList<T>::DoublyLinkedList() : head(nullptr), tail(nullptr) {}

template<typename T>

void DoublyLinkedList<T>::print() {

Node2<T>\* current = head;

while (current) {

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

current = current->next;

}

cout << endl;

}

template<typename T>

void DoublyLinkedList<T>::push\_front(T value) {

Node2<T>\* newNode = new Node<T>(value);

if (head == nullptr) {

head = tail = newNode;

} else {

newNode->next = head;

head->prev = newNode;

head = newNode;

}

size++;

}

template<typename T>

void DoublyLinkedList<T>::push\_back(T value) {

Node2<T>\* newNode = new Node<T>(value);

if (tail == nullptr) {

head = tail = newNode;

} else {

newNode->prev = tail;

tail->next = newNode;

tail = newNode;

}

size++;

}

template<typename T>

void DoublyLinkedList<T>::pop\_front() {

if (head == nullptr) return;

Node2<T>\* temp = head;

head = head->next;

if (head != nullptr) {

head->prev = nullptr;

} else {

tail = nullptr; // Список стал пустым

}

delete temp;

size--;

}

template<typename T>

void DoublyLinkedList<T>::pop\_back() {

if (tail == nullptr) return; // Список пуст

Node2<T>\* temp = tail;

tail = tail->prev;

if (tail != nullptr) {

tail->next = nullptr;

} else {

head = nullptr; // Список стал пустым

}

delete temp;

size--;

}

template<typename T>

void DoublyLinkedList<T>::remove(T value) {

Node2<T>\* current = head;

while (current) {

if (current->data == value) {

if (current->prev) {

current->prev->next = current->next;

} else {

head = current->next; // Удаляем голову

}

if (current->next) {

current->next->prev = current->prev;

} else {

tail = current->prev; // Удаляем хвост

}

delete current;

return; // Выход после удаления первого найденного элемента

}

current = current->next;

}

size--;

}

template<typename T>

bool DoublyLinkedList<T>::find(T value) {

Node2<T>\* current = head;

while (current) {

if (current->data == value) {

return true; // Элемент найден

}

current = current->next;

}

return false; // Элемент не найден

}

template<typename T>

DoublyLinkedList<T>::~DoublyLinkedList() {

while (head) {

pop\_front();

}

}

# Вывод

В ходе выполнения практической работы №1 я познакомился с понятием СУБД, SQL базы данных. Научился реализовывать свою собственную реляционную СУБД при поддержки некоторых запросов на языке SQL.

Ссылка на репозиторий: [нажмите](https://github.com/JamesSevil/2kurs-pract1)