Федеральное государственное образовательное учреждение

высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

КАФЕДРА СИСТЕМНОГО АНАЛИЗА И ИНТЕГРАЦИИ IT-РЕШЕНИЙ

**ПРОВЕРИЛ ВЫПОЛНИЛ**

Старший преподаватель кафедры САИТ Студент группы КТбо1-2

Лапшин Вячеслав Сергеевич Хиль Дмитрий Сергеевич

«01» апреля 2023 г. «06» апреля 2023 г.

**ЛАБОРАТОРНАЯ РАБОТА №3**

По дисциплине

«Основы алгоритмизации и

программирования»

На тему

«Работа со стеком»

Вариант 5

**Таганрог 2023**

**Цель**: Цель данной лабораторной работы состоит в том, чтобы научить студентов работать со структурами данных и работой со стеком.

**Задачи**: Написать программы для работы с односвязным и двусвязным списками в соответствии с выданным вариантом задания. Предусмотреть в программах следующие функции:

1. Включить новый элемент в конец списка.

2. Включить новый элемент на заданное пользователем место списка.

3. Включить новый элемент после элемента с заданной информационной частью.

4. Включить новый элемент перед элементом с заданной информационной частью.

5. Включить новый элемент в середину списка.

6. Исключить элемент из середины списка.

7. Исключить элемент с заданной информационной частью.

8. Исключить элемент из конца списка.

9. Исключить элемент из заданного пользователем места списка.

10. Исключить элемент из головы списка.

**Ход работы**: Телефонная книга. Организовать список с названием Person и обращаться к нему, все данные считываются с файла.

**Проектирование:**

Изображение выглядит как диаграмма

Автоматически созданное описание

Рисунок 1. – Функция main.

**Реализация:**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Person {

string Phone;

Person\* next;

Person\* prev;

};

Person\* head = nullptr;

Person\* tail = nullptr;

void display(Person\* p) {

cout << "Phone: " << p->Phone << endl;

}

void displayAll() {

if (head == nullptr) {

cout << "Address book is empty" << endl;

return;

}

Person\* curr = head;

while (curr != nullptr) {

display(curr);

cout << endl;

curr = curr->next;

}

}

void insertAtEnd(string Phone) {

Person\* p = new Person;

p->Phone = Phone;

p->next = nullptr;

p->prev = tail;

if (tail != nullptr) {

tail->next = p;

}

tail = p;

if (head == nullptr) {

head = p;

}

}

void insertAtPos(int pos, string Phone) {

Person\* p = new Person;

p->Phone = Phone;

p->next = nullptr;

p->prev = nullptr;

if (pos == 1) {

p->next = head;

head->prev = p;

head = p;

return;

}

Person\* curr = head;

for (int i = 1; i < pos - 1 && curr != nullptr; i++) {

curr = curr->next;

}

if (curr == nullptr) {

cout << "Invalid position" << endl;

return;

}

p->prev = curr;

p->next = curr->next;

if (curr->next != nullptr) {

curr->next->prev = p;

}

curr->next = p;

}

void insertAfter(string Phone, string newPhone) {

Person\* curr = head;

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

curr = curr->next;

}

if (curr == nullptr) {

cout << "Person not found" << endl;

return;

}

Person\* p = new Person;

p->Phone = newPhone;

p->next = curr->next;

p->prev = curr;

if (curr->next != nullptr) {

curr->next->prev = p;

}

curr->next = p;

if (tail == curr) {

tail = p;

}

}

void insertBefore(string Phone, string newPhone) {

Person\* curr = head;

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

curr = curr->next;

}

if (curr == nullptr) {

cout << "Person not found" << endl;

return;

}

Person\* p = new Person;

p->Phone = newPhone;

p->prev = curr->prev;

p->next = curr;

if (curr->prev != nullptr) {

curr->prev->next = p;

}

curr->prev = p;

if (head == curr) {

head = p;

}

}

void insertInMiddle(string Phone) {

Person\* slow = head;

Person\* fast = head;

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

slow = slow->next;

fast = fast->next->next;

}

Person\* p = new Person;

p->Phone = Phone;

p->prev = slow;

p->next = slow->next;

if (slow->next != nullptr) {

slow->next->prev = p;

}

slow->next = p;

if (tail == slow) {

tail = p;

}

}

void deleteFromMiddle() {

if (head == nullptr) {

cout << "Address book is empty" << endl;

return;

}

Person\* slow = head;

Person\* fast = head;

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

slow = slow->next;

fast = fast->next->next;

}

if (slow == head) {

cout << slow << endl;

cout << head << endl;

head = head->next;

if (head != nullptr) {

head->prev = nullptr;

}

else {

tail = nullptr;

}

delete slow;

}

else if (slow == tail) {

cout << slow << endl;

cout << tail << endl;

tail = tail->prev;

if (tail != nullptr) {

tail->next = nullptr;

}

else {

head = nullptr;

}

delete slow;

}

else {

slow->prev->next = slow->next;

slow->next->prev = slow->prev;

delete slow;

}

}

void deleteByPhone(Person\*\* head, const std::string& Phone) {

if (\*head == nullptr) {

std::cout << "List is empty. Cannot delete element.\n";

return;

}

Person\* curr = \*head;

Person\* prev = nullptr;

while (curr != nullptr) {

if (curr->Phone == Phone) {

if (prev == nullptr) {

\*head = curr->next;

}

else {

prev->next = curr->next;

}

delete curr;

std::cout << "Element with Phone " << Phone << " deleted.\n";

return;

}

prev = curr;

curr = curr->next;

}

std::cout << "Element with Phone " << Phone << " not found in list.\n";

}

void deleteByInfo(string Phone) {

Person\* curr = head;

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

curr = curr->next;

}

if (curr == nullptr) {

cout << "Person not found" << endl;

return;

}

if (curr == head) {

head = head->next;

if (head != nullptr) {

head->prev = nullptr;

}

else {

tail = nullptr;

}

delete curr;

}

else if (curr == tail) {

tail = tail->prev;

if (tail != nullptr) {

tail->next = nullptr;

}

else {

head = nullptr;

}

delete curr;

}

else {

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

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

delete curr;

}

}

void deleteFromEnd() {

if (head == nullptr) {

cout << "Address book is empty" << endl;

return;

}

if (tail == head) {

delete head;

head = nullptr;

tail = nullptr;

return;

}

Person\* curr = tail;

tail = tail->prev;

tail->next = nullptr;

delete curr;

}

void deleteFromPos(int pos) {

if (head == nullptr) {

cout << "Address book is empty" << endl;

return;

}

if (pos == 1) {

return;

}

Person\* curr = head;

int i = 1;

while (i < pos && curr != nullptr) {

curr = curr->next;

i++;

}

if (curr == nullptr) {

cout << "Position not found" << endl;

return;

}

if (curr == tail) {

tail = tail->prev;

if (tail != nullptr) {

tail->next = nullptr;

}

else {

head = nullptr;

}

delete curr;

}

else {

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

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

delete curr;

}

}

void deleteFromBeginning() {

if (head == nullptr) {

cout << "Address book is empty" << endl;

return;

}

if (head == tail) {

delete head;

head = nullptr;

tail = nullptr;

return;

}

Person\* curr = head;

head = head->next;

head->prev = nullptr;

delete curr;

}

void display() {

Person\* curr = head;

while (curr != nullptr) {

cout << "Phone: " << curr->Phone << endl;

cout << endl;

curr = curr->next;

}

}

void saveToFile() {

ofstream outFile("file.txt");

Person\* curr = head;

while (curr != nullptr) {

outFile << curr->Phone << endl;

curr = curr->next;

}

outFile.close();

}

void loadFromFile() {

ifstream inFile("file.txt");

if (!inFile) {

cout << "Unable to open file" << endl;

return;

}

string Phone;

while (getline(inFile, Phone)) {

insertAtEnd(Phone);

}

inFile.close();

}

int main() {

loadFromFile();

string Phone;

string afterPhone, beforePhone;

while (true) {

int choice;

cout << "Enter your choice:" << endl;

cout << "1. Add a new person at the end of the list" << endl;

cout << "2. Add a new person at possition" << endl;

cout << "3. Add a new person after a person with a specified Phone" << endl;

cout << "4. Add a new person before a person with a specified Phone" << endl;

cout << "5. Add a new person in the middle of the list" << endl;

cout << "6. Delete a person from the middle of the list" << endl;

cout << "8. Delete a person from the end of the list" << endl;

cout << "9. Delete a person from a user-defined location in the list" << endl;

cout << "10. Delete a person from the beginning of the list" << endl;

cout << "11. Display the address book" << endl;

cout << "12. Exit" << endl;

cout << "PLEASE, NOT USE A SPACE BAR, USE '\_' in Phone:\nExample: Vasiliy\_Vaska" << endl;

cin >> choice;

switch (choice) {

case 1:

cout << "Enter Phone:\n";

cin >> Phone;

insertAtEnd(Phone);

saveToFile();

break;

case 2:

int pos2;

cout << "Enter position:\n";

cin >> pos2;

cout << "Enter new Phone: ";

cin >> Phone;

insertAtPos(pos2, Phone);

saveToFile();

break;

case 3:

cout << "Enter Phone:\n";

cin >> Phone;

cout << "Enter Phone after which to insert: ";

cin >> afterPhone;

insertAfter(Phone, afterPhone);

saveToFile();

break;

case 4:

cout << "Enter Phone:\n";

cin >> Phone;

cout << "Enter Phone before which to insert: ";

cin >> beforePhone;

insertBefore(Phone, beforePhone);

saveToFile();

break;

case 5:

cout << "Enter Phone:\n";

cin >> Phone;

insertInMiddle(Phone);

saveToFile();

break;

case 6:

cout << "Enter position of person to delete: ";

deleteFromMiddle();

saveToFile();

case 8:

deleteFromEnd();

saveToFile();

break;

case 9:

cout << "Enter position of person to delete: ";

int pos1;

cin >> pos1;

deleteFromPos(pos1);

saveToFile();

break;

case 10:

deleteFromBeginning();

saveToFile();

break;

case 11:

display();

break;

case 12:

return 0;

default:

cout << "Invalid choice" << endl;

}

}

return 0;

}

**Вывод:** в ходе лабораторной работы я освоил способы представления и обработки данных в виде списков. Улучшил свои умения работы со стеком и реализовал односвязный и двусвязный список, по данному заданию.