Федеральное государственное автономное образовательное учреждение

высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

КАФЕДРА СИСТЕМНОГО АНАЛИЗА И ТЕЛЕКОММУНИКАЦИИ

**ПРОВЕРИЛ ВЫПОЛНИЛ**

Старший преподаватель кафедры САиТ Студент группы КТбо1-2

Лапшин В.С. Погосян А.С.

«27» Марта 2023 г «23» Марта 2023 г.

**ЛАБОРАТОРНАЯ РАБОТА №3**

По дисциплине

«Основы алгоритмизации и программирования»

Вариант 8

На тему

«Работа со стэком»

**Таганрог 2023**

**Цель**: Цель данной лабораторной работы состоит в том, чтобы научить студентов работать со сложными типами данных, каковыми являются структуры данных.

**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.

Вариант задания:

**Ход работы**: Адресная книга. Организовать список с названием Person и обращаться к нему, все данные считываются с файла и записываются в вектор пар для удобного использования.

**Проектирование:**

Изображение выглядит как диаграмма

Автоматически созданное описание

*Рисунок 1 – блок-схема работы программы*

**Реализация .**

*Программа выполнена на языке С, с использованием некоторых возможностей С++ (без ООП и высокоуровневых библиотек)*

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <string>  using namespace std;  struct Person {  string name;  string address;  string phone;  Person\* next;  Person\* prev;  };  Person\* head = nullptr;  Person\* tail = nullptr;  void display(Person\* p) {  cout << "Name: " << p->name << endl;  cout << "Address: " << p->address << endl;  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 name, string address, string phone) {  Person\* p = new Person;  p->name = name;  p->address = address;  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 name, string address, string phone) {  Person\* p = new Person;  p->name = name;  p->address = address;  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 name, string newName, string newAddress, string newPhone) {  Person\* curr = head;  while (curr != nullptr && curr->name != name) {  curr = curr->next;  }  if (curr == nullptr) {  cout << "Person not found" << endl;  return;  }  Person\* p = new Person;  p->name = newName;  p->address = newAddress;  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 name, string newName, string newAddress, string newPhone) {  Person\* curr = head;  while (curr != nullptr && curr->name != name) {  curr = curr->next;  }  if (curr == nullptr) {  cout << "Person not found" << endl;  return;  }  Person\* p = new Person;  p->name = newName;  p->address = newAddress;  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 name, string address, 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->name = name;  p->address = address;  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) {  head = head->next;  if (head != nullptr) {  head->prev = nullptr;  }  else {  tail = nullptr;  }  delete slow;  }  else if (slow == tail) {  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 deleteByName(Person\*\* head, const std::string& name) {  if (\*head == nullptr) {  std::cout << "List is empty. Cannot delete element.\n";  return;  }  Person\* curr = \*head;  Person\* prev = nullptr;  while (curr != nullptr) {  if (curr->name == name) {  if (prev == nullptr) {  \*head = curr->next;  }  else {  prev->next = curr->next;  }  delete curr;  std::cout << "Element with name " << name << " deleted.\n";  return;  }  prev = curr;  curr = curr->next;  }  std::cout << "Element with name " << name << " not found in list.\n";  }  void deleteByInfo(string name) {  Person\* curr = head;  while (curr != nullptr && curr->name != name) {  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) {  delete head;  head = nullptr;  if (head != nullptr) {  head->prev = nullptr;  }  else {  tail = nullptr;  }  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 == 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 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 << "Name: " << curr->name << endl;  cout << "Address: " << curr->address << endl;  cout << "Phone: " << curr->phone << endl;  cout << endl;  curr = curr->next;  }  }  void saveToFile() {  ofstream outFile("file.txt");  Person\* curr = head;  while (curr != nullptr) {  outFile << curr->name << endl;  outFile << curr->address << endl;  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 name, address, phone;  while (getline(inFile, name)) {  getline(inFile, address);  getline(inFile, phone);  insertAtEnd(name, address, phone);  }  inFile.close();  }  int main() {  loadFromFile();    string name, address, phone;  string afterName, beforeName;  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 a user-defined location in the list" << endl;  cout << "3. Add a new person after a person with a specified name" << endl;  cout << "4. Add a new person before a person with a specified name" << 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 << "7. Delete a person with a specified name" << 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. Save the address book to a file" << endl;  cout << "13. Exit" << endl;  cin >> choice;  switch (choice) {  case 1:  cout << "Enter name, address, and phone number: \nPLEASE, NOT USE A SPACE BAR, USE '\_' \nExample: Vanya, Main\_st\_12, 88005553535\n\n Data: ";  cin >> name >> address >> phone;  insertAtEnd(name, address, phone);  saveToFile();  break;  /\*case 2:  cout << "Enter name, address, and phone number: ";  cin >> name >> address >> phone;  int pos;  cout << "Enter position: ";  cin >> pos;  insertAtPos(name, address, phone, pos);  break;\*/  case 3:  cout << "Enter name, address, and phone number: ";  cin >> name >> address >> phone;  cout << "Enter name after which to insert: ";    cin >> afterName;  insertAfter(name, address, phone, afterName);  saveToFile();  break;  case 4:  cout << "Enter name, address, and phone number: ";  cin >> name >> address >> phone;  cout << "Enter name before which to insert: ";  cin >> beforeName;  insertBefore(name, address, phone, beforeName);  saveToFile();  break;  case 5:  cout << "Enter name, address, and phone number: ";  cin >> name >> address >> phone;  insertInMiddle(name, address, phone);  saveToFile();  break;  case 6:  cout << "Enter position of person to delete: ";  int pos;  cin >> pos;  deleteFromPos(pos);  saveToFile();  break;  /\*case 7: //Ошибка стринг в Персон  cout << "Enter name of person to delete: ";  cin >> name;  deleteByName(name);  break;\*/  case 8:  deleteFromEnd();  saveToFile();  break;  case 9:  cout << "Enter position of person to delete: ";  cin >> pos;  deleteFromPos(pos);  saveToFile();  break;  case 10:  deleteFromBeginning();  saveToFile();  break;  case 11:  display();  break;  case 12:  saveToFile();  break;  case 13:  return 0;  default:  cout << "Invalid choice" << endl;  }  }  return 0;  } |

**Результат работы программы.**

*На данных скриншотах можно наблюдать работу программы*

*Изображение выглядит как текст, монитор, в помещении, снимок экрана

Автоматически созданное описание*

*Рисунок 2 – результат работы программы*

**Вывод:** В этой лабораторной работе мы реализовали структуры данных в виде одно- и двусвязных списков на C++ для хранения и манипулирования информацией о людях, включая их имя, адрес и номер телефона. Мы также реализовали различные функции для добавления, удаления и изменения узлов в связанных списках.

Использование связанных списков в качестве структуры данных дает несколько преимуществ по сравнению с другими структурами данных, такими как массивы или векторы. Связанные списки могут динамически выделять память для новых узлов, позволяя списку увеличиваться или уменьшаться по мере необходимости. Кроме того, связанные списки можно использовать для реализации более сложных структур данных, таких как очереди или стеки, а также для эффективного выполнения определенных операций, таких как вставка или удаление узлов.

Благодаря этой лабораторной работе мы приобрели ценный опыт работы со связанными списками и научились использовать их для решения различных задач. Мы также приобрели опыт реализации функций для манипулирования связанным списком, что является ценным навыком, который может быть применен в других проектах по программированию.

В заключение следует отметить, что реализация одно- и двусвязных списков в C++ обеспечивает мощную и гибкую структуру данных для хранения и манипулирования информацией. Понимая принципы работы связанных списков и функции, необходимые для работы с ними, мы можем создавать более эффективные и действенные программы.