Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

**Алгоритмізація та програмування 2: [Процедурне програмування](https://classroom.google.com/u/1/c/NDQ5MDIwMzkxMzg1" \t "https://classroom.google.com/u/1/c/NDQ5MDIwMzkxMzg1/a/NDYwMzEwMzE1MTE4/_self)**

ЗВІТ

до лабораторної роботи №3

«**Списки**»

*(ТЕМА)*

Варіант № 1

Дата «6» травня 2022 Виконав: студент 1 курсу

гр. ТР-15

Руденко Владислав Ігорович

*(П.І.Б.)*

Оцінка «\_\_\_\_\_\_\_\_\_» Перевірив:

Крячок Олександр Степанович

Дата «\_\_\_» \_\_\_\_\_\_\_ 20\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(П.І.Б., підпис)*

Київ – 2022

***Завдання до роботи:***

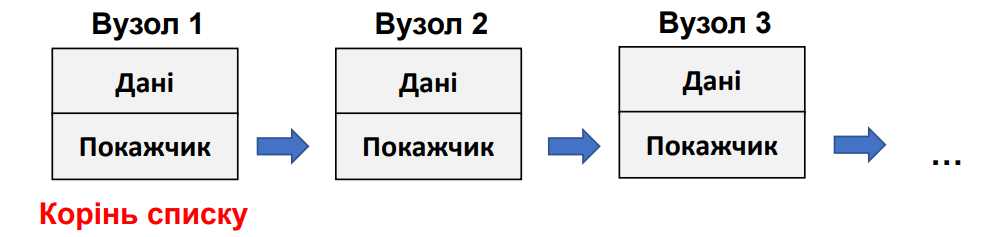
1. Дослідити особливості створення одно- та дво-направлених списків.  
   2. Вивчити і реалізувати механізми додавання нових записів у список,  пошуку записів у списку за певними полями, видалення записів зі списку та редагування знайдених записів, а також збереження всього списку у файлі та зчитування списку із файлу до пам’яті з відновленням всіх зв’язків.  
   3. Розробити Блок-схему програмного алгоритму.  
   4. Оформити ЗВІТ до лабораторної роботи згідно вимог та методичних рекомендацій.



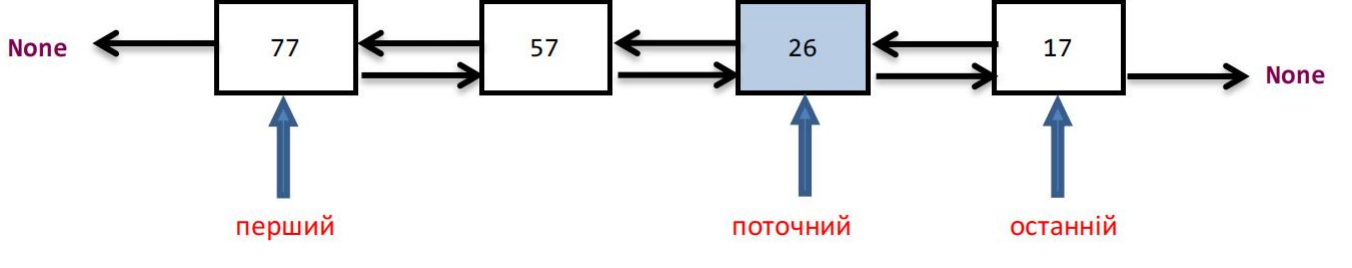
Додаток 4

***Теоритичні відомості:.***Зв’язний список є найпростішим типом даних динамічної структури, що складається з елементів (вузлів). Кожен вузол включає в себе в класичному варіанті два поля:

• дані (в якості даних може виступати змінна, об'єкт класу або структури і т. д.)

• покажчик на наступний вузол в списку  


*Лiнiйний список –* це набiр однотипних компонентiв, якi послiдовно пов’язанi мiж собою за допомогою покажчикiв. Кожен компонент списку може складатися iз кiлькох iнформацiйних полiв та покажчика на наступний елемент. Основні операції: ü пошук елемента з заданими властивостями; ü визначення i-того елемента; ü додавання елемента до, або після вказаного; ü вилучення певного елемента зі списку; ü впорядкування *Двобічно зв’язаний (двозв’язний) список* – динамічна структура даних, що складається з елементів одного типу, зв'язаних між собою у строго визначеному порядку. При цьому визначено перший та останній елементи у списку, а кожен елемент списку вказує на наступний і попередній елементи у списку. Перший лемент має попереднім елементом по силання на невизначений елемент None. Аналогічно, None буде наступним елементом для останнього елементу списк



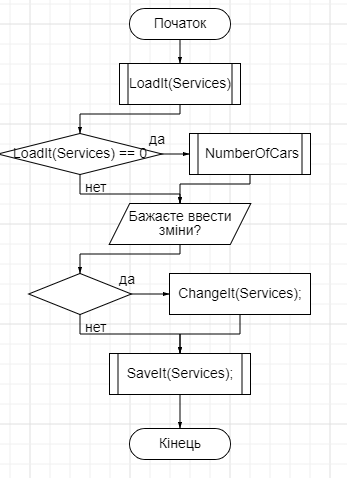
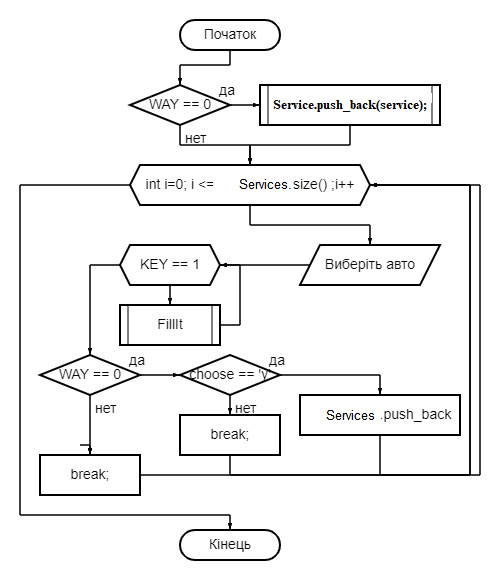
Базовий набір дій над двозв’язними списками:

1. Створити список.
2. Операція визначення, чи порожній список. Повертає булеве значення.
3. Зробити поточними перший елемент списку.
4. Зробити поточними останній елемент списку.
5. Перейти до наступного елемента. 6. Перейти до попереднього елемента.
6. Отримати поточний елемент.
7. Вставити новий елемент перед поточним.
8. Вставити новий елемент після поточного.
9. Видалити

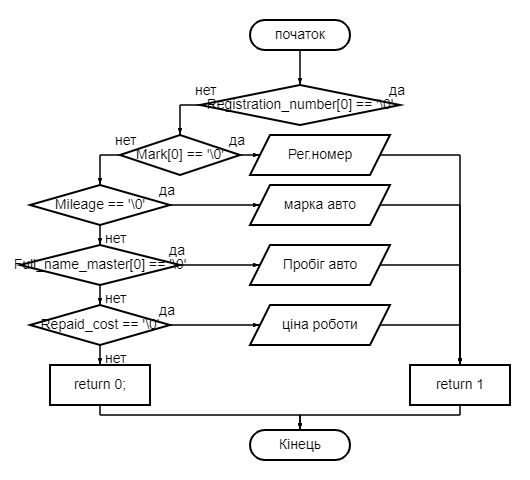
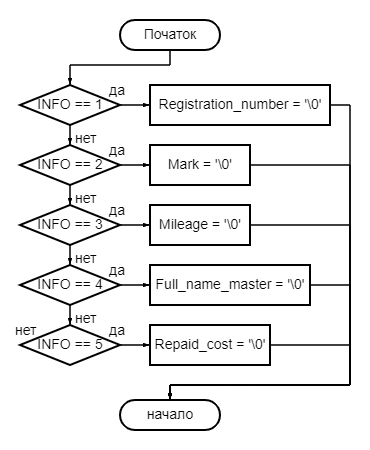
***Опис програми:***

*Перед початком виконання операцій з списком програма перевіряє чи наявний файл необхідного формату та при його наявносіт зчитує елементи списку з документа та передає їх списку, при їх відсутності оператор власноруч вводить дані. Пілся вводу оператору надається можливість відредагувати введені значення в списку шляхом їх видалення та подальшого перезаповнення вибраного елементу. Елементи опрацювання списку були виведені в окремий файл “List.cpp”та приєднані до основного. Після редагування програма записує дані структури з списку до вихідного файлу.*

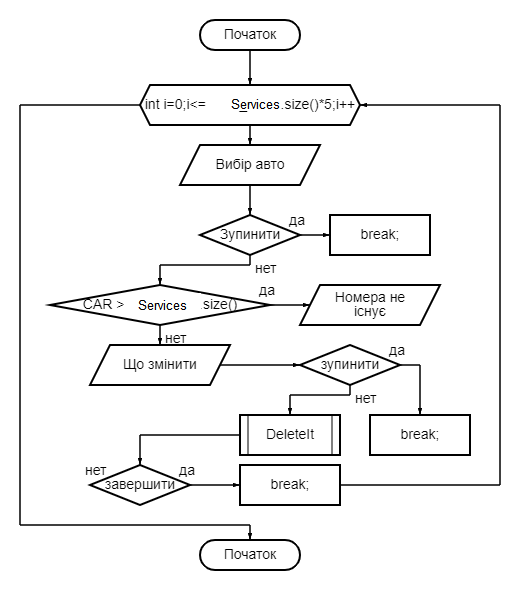
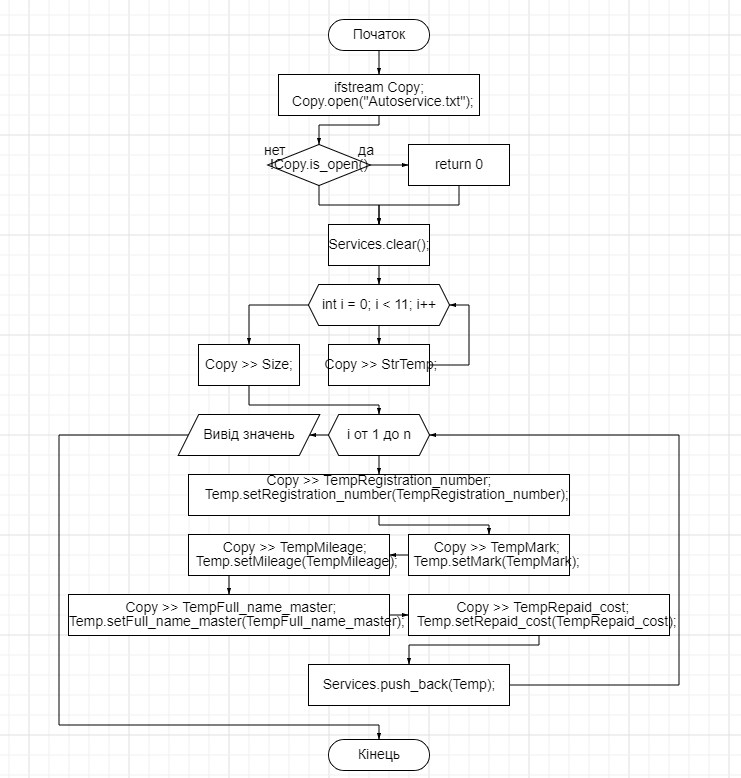
***Блок схема алгоритму:***

Блок-схема 1 (main) Блок-схема 2 (NumberOfCars)

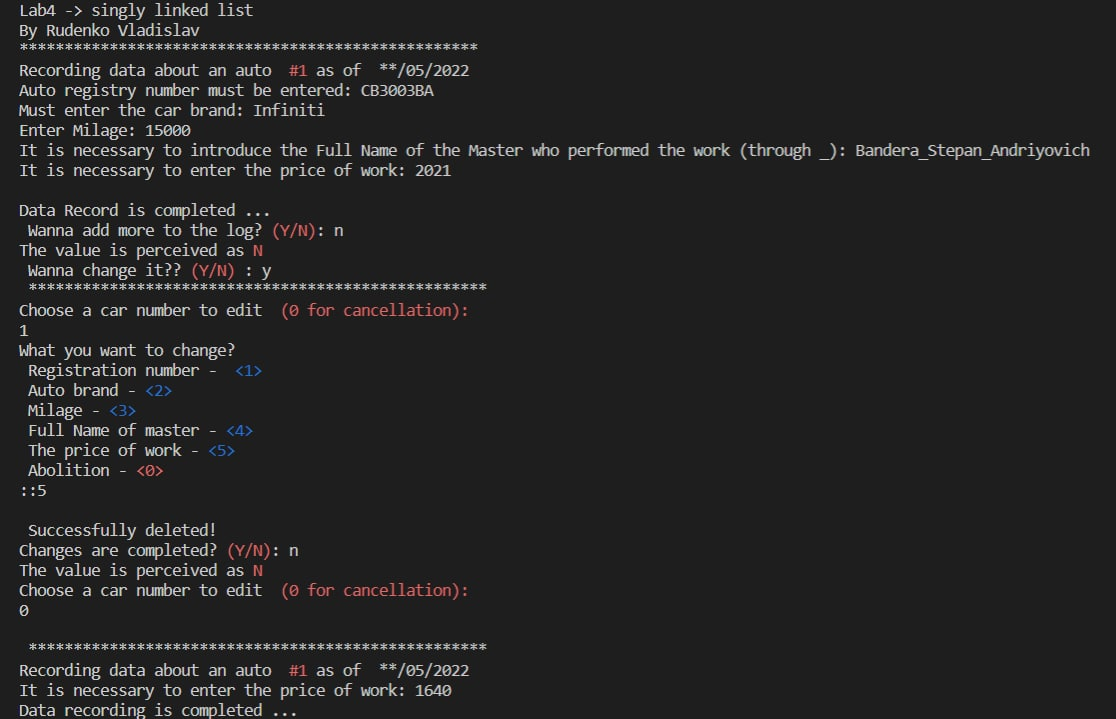
****** 

*Блок-схема 3 (FillIt) Блок-схема 4 (DeleteIt)*

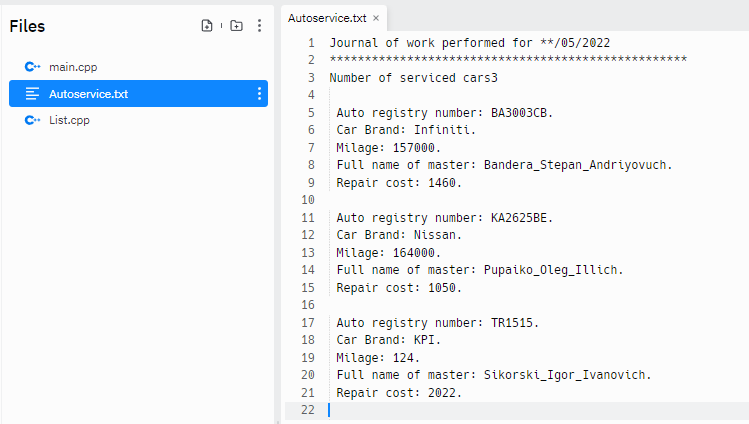
****** 

*Блок-схема 5 (ChangeIt) Блок-схема 4 (LoadIt)*

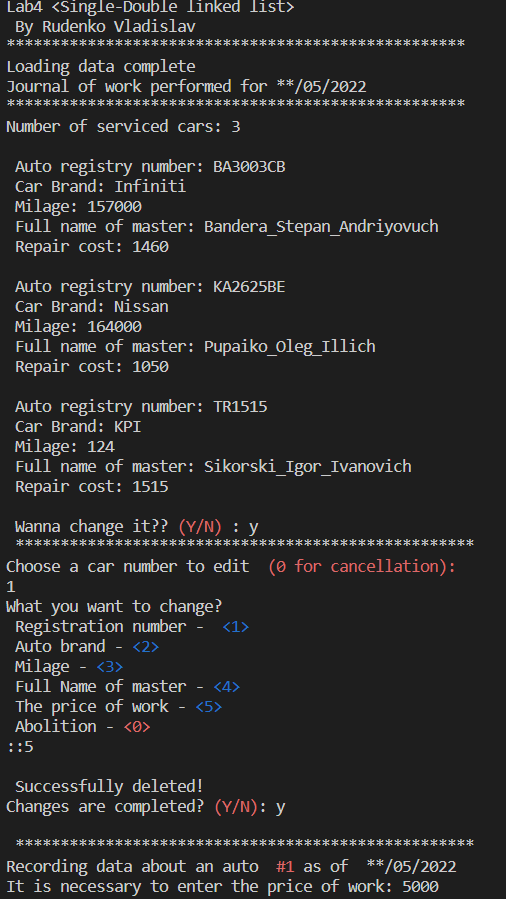
***Результати роботи:***



Результат завдання 1 (Виконання в консолі)



Результат завдання 2 (Збережені дані)

  
Результат завдання 3 (Завантаження збережених даних)

***Посилання на програму в repl.it:***

***<https://replit.com/join/eabxdfvbwq-hetik>***

***Висновки:***

В ході виконання Лабораторної роботи №4 було розглянуто особливості створення та використання одно- та дво-направлених списків. Результатом лабораторної роботи №4 є програма, яка використовує значення з файлу (якщо відсутні то вводиться самостійно) та записує їх до дво-направленого списку. Реалізовано пошук елементів за індексом, видалення та запис окремих елементів списку, а також збереження усього списку до файлу. Розроблено блок-схему основного алгоритму. Програма має можливість редагувати елементи структури, та заміняти частини їх в списку. Знайдено та проаналізовано деяку кількість помилок.

**Програмний код (у текстовому варіанті):*****Main.cpp:***

#include <iostream>

#include <vector>

#include <string>

#include "List.cpp"

#include <clocale>

#include <fstream>

#define RED "\x1b[31m"

#define BLUE "\x1b[34m"

#define PINK "\033[1;35m"

#define RESET "\x1b[0m"

using namespace std;

struct Autoservice

{

Autoservice(string TRegistration\_number = "0", string TMark = "0", int TMileage = 0, string TFull\_name\_master = "0", float TRepaid\_cost = 0)

: Registration\_number(TRegistration\_number), Mark(TMark), Mileage(TMileage), Full\_name\_master(TFull\_name\_master), Repaid\_cost(TRepaid\_cost)

{}

public:

//SET//

void setRegistration\_number(string TRegistration\_number){Registration\_number = TRegistration\_number;}

void setMark(string TMark){Mark = TMark;}

void setMileage(int TMileage){Mileage = TMileage;}

void setFull\_name\_master(string TFull\_name\_master){Full\_name\_master = TFull\_name\_master;}

void setRepaid\_cost(float TRepaid\_cost){Repaid\_cost = TRepaid\_cost;}

//GET//

string getRegistration\_number() const {return Registration\_number;}

string getMark() const {return Mark;}

int getMileage() const {return Mileage;}

string getFull\_name\_master() const {return Full\_name\_master;}

float getRepaid\_cost() const {return Repaid\_cost;}

//delete//

void delRegistration\_number(){Registration\_number = "0";}

void delMark(){Mark = "0";}

void delMileage(){Mileage = 0;}

void delFull\_name\_master(){Full\_name\_master = "0";}

void delRepaid\_cost(){Repaid\_cost = 0;}

private:

string Registration\_number;

string Mark;

int Mileage;

string Full\_name\_master;

float Repaid\_cost;

}service;

/////////////////////////////////////////////////////////////////

int FillIt(List<Autoservice>& Services, int

counter);

void NumberOfCars(List<Autoservice>& Services, int WAY);

void DeleteIt(List<Autoservice>& Services, int CAR, int INFO);

void ChangeIt(List<Autoservice>& Services);

void SaveIt(List<Autoservice>& Services);

int LoadIt(List<Autoservice>& Services);

/////////////////////////////////////////////////////////////////

int main()

{

char choose;

cout << "Lab4 <Single-Double linked list>" << "\n By Rudenko Vladislav" << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

List<Autoservice> Services;

if(LoadIt(Services) == 0)

NumberOfCars(Services, 0);

cout << "\n Wanna change it??" RED " (Y/N) " RESET ": ";

cin >> choose;

if (choose == 'Y' || choose == 'y')

ChangeIt(Services);

else if (choose != 'N' || choose != 'n')

cout << "The value is perceived as"<<RED<<" N"<< RESET << endl;

cout << "\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \nExecution is completed.";

SaveIt(Services);

}

int FillIt(List<Autoservice>& Services, int

counter)

{

int KEY;

string temp;

if(Services[counter].getRegistration\_number() == "0")

{

cout << "Auto registry number must be entered: ";

cin >> temp;

Services[counter].setRegistration\_number(temp);

return 1;

}

else if(Services[counter].getMark() == "0")

{

cout << "Must enter the car brand: ";

cin >> temp;

Services[counter].setMark(temp);

return 1;

}

else if(Services[counter].getMileage() == 0)

{

cout << "Enter Milage: ";

cin >> temp;

Services[counter].setMileage(stoi(temp));

return 1;

}

else if(Services[counter].getFull\_name\_master() == "0")

{

cout << "It is necessary to introduce the Full Name of the Master who performed the work (through \_): ";

cin >> temp;

Services[counter].setFull\_name\_master(temp);

return 1;

}

else if(Services[counter].getRepaid\_cost() == 0)

{

cout << "It is necessary to enter the price of work: ";

cin >> temp;

Services[counter].setRepaid\_cost(stod(temp));

return 1;

}

else

return 0;

}

void NumberOfCars(List<Autoservice>& Service, int WAY)

{

if(WAY == 0)

Service.push\_back(service);

for(int i=0; i <= Service.GetSize() ;i++)

{

int lock = 0, KEY = 1;

char choose;

cout << "Recording data about an auto "<< RED <<" #" << i+1 << RESET << " as of "<< RESET <<" \*\*/05/2022\n";

while(KEY == 1)

KEY = FillIt(Service, i);

KEY = 1;

if (WAY == 0)

{

cout << "\nData Record is completed ...\n Wanna add more to the log? "<< RED <<"(Y/N)" << RESET <<": ";

cin >> choose;

if (choose == 'Y' || choose == 'y')

{

Service.push\_back(service);

cout << endl;

}

else if (choose != 'N' || choose != 'n')

{

cout << "The value is perceived as"<<RED<<" N"<< RESET << endl;

break;

}

}

else if(WAY == 1 && Service.GetSize()-1 == i)

break;

}

}

void SaveIt(List<Autoservice>& Services)

{

ofstream Copy;

Copy.open("Autoservice.txt");

Copy << "Journal of work performed for \*\*/05/2022\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nNumber of serviced cars: " << Services.GetSize() << endl;

for(int i=0;i<Services.GetSize();i++)

Copy << "\n Auto registry number: " << Services[i].getRegistration\_number() << "\n Car Brand: " << Services[i].getMark() << "\n Milage: " << Services[i].getMileage() << "\n Full name of master: " << Services[i].getFull\_name\_master() << "\n Repair cost: " << Services[i].getRepaid\_cost() <<endl;;

Copy.close();

Services.clear();

}

int LoadIt(List<Autoservice>& Services)

{

ifstream Copy;

Copy.open("Autoservice.txt");

if(!Copy.is\_open()) return 0;

Services.clear();

Autoservice Temp;

string StrTemp;

for(int i = 0; i < 11; i++)

{

Copy >> StrTemp;

}

int Size = 0;

Copy >> Size;

string TempRegistration\_number;

string TempMark;

int TempMileage;

string TempFull\_name\_master;

float TempRepaid\_cost;

for(int i = 0; i < Size; i++)

{

Copy >> StrTemp >> StrTemp >> StrTemp;

Copy >> TempRegistration\_number;

Temp.setRegistration\_number(TempRegistration\_number);

Copy >> StrTemp >> StrTemp;

Copy >> TempMark;

Temp.setMark(TempMark);

Copy >> StrTemp;

Copy >> TempMileage;

Temp.setMileage(TempMileage);

Copy >> StrTemp >> StrTemp >> StrTemp >> StrTemp;

Copy >> TempFull\_name\_master;

Temp.setFull\_name\_master(TempFull\_name\_master);

Copy >> StrTemp >> StrTemp;

Copy >> TempRepaid\_cost;

Temp.setRepaid\_cost(TempRepaid\_cost);

Services.push\_back(Temp);

}

cout << "Loading data complete\n";

cout << "Journal of work performed for \*\*/05/2022\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nNumber of serviced cars: " << Services.GetSize() << endl;

for(int i=0;i<Services.GetSize();i++)

cout << "\n Auto registry number: " << Services[i].getRegistration\_number() << "\n Car Brand: " << Services[i].getMark() << "\n Milage: " << Services[i].getMileage() << "\n Full name of master: " << Services[i].getFull\_name\_master() << "\n Repair cost: " << Services[i].getRepaid\_cost() <<endl;;

Copy.close();

return 1;

}

void DeleteIt(List<Autoservice>& Services, int CAR, int INFO)

{

switch(INFO)

{

case 1:

Services[CAR].delRegistration\_number();

break;

case 2:

Services[CAR].delMark();

break;

case 3:

Services[CAR].delMileage();

break;

case 4:

Services[CAR].delFull\_name\_master();

break;

case 5:

Services[CAR].delRepaid\_cost();

break;

}

cout << "\n Successfully deleted!\n";

}

void ChangeIt(List<Autoservice>& Services)

{

char choose;

int CAR, INFO;

vector <int> NeedChange;

cout << " \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n";

for(int i=0;i<= Services.GetSize()\*5;i++)

{

begin:

cout << "Choose a car number to edit " RED " (0 for cancellation): " RESET << endl;

cin >> CAR;

if(CAR == 0)

break;

else

if(CAR > Services.GetSize() || CAR<0){

cout << RED "A given car does not inspire" RESET << endl;

goto begin;

}

else

{

cout << "What you want to change?\n Registration number - " BLUE " <1> " RESET "\n Auto brand -" BLUE " <2>" RESET "\n Milage -" BLUE " <3>" RESET "\n Full Name of master -" BLUE " <4>" RESET "\n The price of work -" BLUE " <5>" RESET "\n Abolition -" RED " <0>" RESET "\n::";

cin >> INFO;

if(INFO == 0)

break;

else

{

DeleteIt(Services, CAR-1, INFO);

NeedChange.push\_back(CAR);

}

}

cout << "Changes are completed?" RED " (Y/N)" RESET ": ";

cin >> choose;

if (choose == 'Y' || choose == 'y')

break;

else if (choose != 'N' || choose != 'n')

cout << "The value is perceived as"<<RED<<" N"<< RESET << endl;

}

cout << "\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n";

NumberOfCars(Services, 1);

NeedChange.clear();

}

***List.spp:***

template<typename T>

class Node

{

    public:

    Node\* pNext;

    Node\* pPerv;

    T Data;

    Node(T Data = T(), Node \*pNext = nullptr, Node \*pPerv = nullptr)

    {

        this->Data = Data;

        this->pNext = pNext;

        this->pPerv = pPerv;

    }

};

template<typename T>

class List

{

public:

    List();

    ~List();

    void push\_back(T Data);

    T& operator[](const int index);

    int GetSize() { return Size; }

    void pop\_front();

    void clear();

    void insert(T data, int index);

    void removeAt(int index);

private:

    int Size;

    Node<T> \*head;

    Node<T> \*tail;

};

template<typename T>

void List<T>::push\_back(T Data)

{

    if(head == nullptr)

    {

        head = new Node<T>(Data);

        tail = head;

    }

    else

    {

      Node<T> \*current = this->head;

      while(current->pNext != nullptr)

      {

          current = current->pNext;

      }

      current ->pNext = new Node<T>(Data, nullptr, current);

      tail = current->pNext;

    }

    Size++;

}

template<typename T>

T & List<T>::operator[](const int index)

{

    int counter = 0;

    Node<T> \*current = this->head;

    while (current != nullptr)

    {

        if (counter == index)

        {

            return current->Data;

        }

        current = current->pNext;

        counter++;

    }

}

template<typename T>

void List<T>::pop\_front()

{

    Node<T> \*temp = head;

    head = head->pNext;

    delete temp;

    Size--;

}

template<typename T>

void List<T>::clear()

{

    while (Size)

    {

        pop\_front();

    }

}

template<typename T>

void List<T>::insert(T data, int index)

{

    if (index == 0)

    {

        push\_back(data);

    }

    else

    {

        Node<T> \*previous = this->head;

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

        {

            previous = previous->pNext;

        }

        Node<T> \*newNode = new Node<T>(data, previous->pNext);

        previous->pNext = newNode;

        Size++;

    }

}

template<typename T>

void List<T>::removeAt(int index)

{

    if (index == 0)

    {

        pop\_front();

    }

    else

    {

        Node<T> \*previous = this->head;

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

        {

            previous = previous->pNext;

        }

        Node<T> \*toDelete = previous->pNext;

        previous->pNext = toDelete->pNext;

        delete toDelete;

        Size--;

    }

}

template<typename T>

List<T>::List()

{

    head = nullptr;

    tail = nullptr;

    Size = 0;

}

template<typename T>

List<T>::~List(){clear();}