**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**  
Федеральное государственное бюджетное образовательное учреждение высшего образования  
**«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА (МИИТ)»  
 РУТ (МИИТ)**  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
Кафедра: «Вычислительные системы и сети»  
  
  
  
  
  
  
  
  
  
  
**О Т Ч Ё Т  
П О Л А Б О Р А Т О Р Н О Й Р А Б О Т Е № 4**  
по дисциплине «Структуры и алгоритмы обработки данных»  
на тему: «Циклические, многомерные и двунаправленные списки»  
  
  
  
  
  
  
  
  
  
 Выполнил: студент группы УИБ-311   
 Владыка П.А.

Москва 2018

**Задание**

Разработать и отладить программу, реализующую обработку списков при связанном распределении. Организовать возможность вставки, удаления и поиска элемента, используя меню выбора.

4. Дек с тремя связями, указывающими на предыдущий, последующий и через один элемент.

**Таблица имён**

|  |  |
| --- | --- |
| **Исходные данные** | |
| Knot | Указатель на первый элемент списка (поле класса USEDECK) |
| **Промежуточные данные** | |
| LastElement | Указатель на прошлый элемент списка |
| Element | Указатель на текущий элемент списка |
| Value | Значение, вводимое с клавиатуры, которое присваивается полю inf элемента списка |
| **Поля структуры ELEMENT** | |
| inf | Информационное поле |
| pLastElement | Указатель на прошлый элемент списка |
| pNewElement | Указатель на следующий элемент списка |
| pOverLastElement | Указатель через один элемент назад |

**Блок-схема**



**Исходный код программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<windows.h>

#include <iomanip>

#include <malloc.h>

using namespace std;

/\* Класс создан для работы со списком с тремя связями \*/

class TRIPLEKNOTE {

public:

template<typename ClassName, typename Type>

inline static void AddRightElement(ClassName\*\* Element, Type value) {

ClassName \*LastElement = \*Element;

\*Element = new ClassName;

(\*Element)->pNewElement = NULL;

if (LastElement != NULL) {

LastElement->pNewElement = \*Element; // Связывание предыдущего элемента со следующим

(\*Element)->pOverLastElement = LastElement->pLastElement; }

else { (\*Element)->pOverLastElement = NULL; }

(\*Element)->inf = value;

(\*Element)->pLastElement = LastElement;// Связывание нового элемента с предыдущим }

template<typename ClassName, typename Type>

static void AddLeftElement(ClassName\*\* Element, Type value) {

if (\*Element != NULL) {

ClassName \*LastElement = \*Element;

for (;;) {

if ((\*Element)->pOverLastElement == NULL) break;

\*Element = (\*Element)->pOverLastElement; }

if ((\*Element)->pLastElement != NULL) {

(\*Element)->pLastElement->pLastElement = new ClassName;

(\*Element)->pLastElement->pLastElement->inf = value;

/\* Редактирование ссылок (всего 4 ссылки) \*/

(\*Element)->pLastElement->pLastElement->pLastElement = NULL;

(\*Element)->pLastElement->pLastElement->pNewElement = (\*Element)->pLastElement;

(\*Element)->pLastElement->pLastElement->pOverLastElement = NULL;

(\*Element)->pOverLastElement = (\*Element)->pLastElement->pLastElement; }

else {

(\*Element)->pLastElement = new ClassName;

(\*Element)->pLastElement->inf = value;

/\* Редактирование ссылок (всего 4 ссылки) \*/

(\*Element)->pLastElement->pLastElement = NULL;

(\*Element)->pLastElement->pNewElement = \*Element;

(\*Element)->pLastElement->pOverLastElement = NULL;

if ((\*Element)->pNewElement != NULL)

(\*Element)->pNewElement->pOverLastElement = (\*Element)->pLastElement; }

\*Element = LastElement; }

else {

\*Element = new ClassName;

(\*Element)->inf = value;

/\* Редактирование ссылок (так как первый элемент, то 3 ссылки)\*/

(\*Element)->pLastElement = NULL;

(\*Element)->pOverLastElement = NULL;

(\*Element)->pNewElement = NULL; }}

template<typename ClassName>

inline static bool DelRightElement(ClassName\*\* Element) {

if (\*Element != NULL) {

ClassName \*LastElement = \*Element;

if ((\*Element)->pLastElement != NULL) {

\*Element = (\*Element)->pLastElement; // Отсылка к левому более старому элементу списка

(\*Element)->pNewElement = NULL; // Удаление, ссылки на удаляемый элемент }

else \*Element = NULL;// Если нет отсылки, обнуление указателя

delete LastElement;

return true; }

return false; }

template<typename ClassName>

static bool DelLeftElement(ClassName\*\* Element) {

if (\*Element != NULL) {

ClassName \*LastElement = \*Element; // Сохранение самого нового элемента в списке

for (;;) {

if ((\*Element)->pOverLastElement == NULL) {

if ((\*Element)->pLastElement == NULL) // Если самый старый элемент в деке {

if ((\*Element)->pNewElement != NULL) {

(\*Element)->pNewElement->pLastElement = NULL;// Редактирование ссылки

if ((\*Element)->pNewElement->pNewElement != NULL) {

(\*Element)->pNewElement->pNewElement->pOverLastElement = NULL;// Редактирование ссылки

delete \*Element;

\*Element = LastElement;// Восстановление указателя на самый новый элемент в деке

break;}}

delete \*Element;

\*Element = NULL;}

else {

delete (\*Element)->pLastElement;

(\*Element)->pLastElement = NULL;// Редактирование ссылки

if ((\*Element)->pNewElement != NULL)

(\*Element)->pNewElement->pOverLastElement = NULL;// Редактирование ссылки

\*Element = LastElement;// Восстановление указателя на самый новый элемент в деке}

break; // Тут выход к завершению с true}

\*Element = (\*Element)->pOverLastElement; // Переход к более старому элементу через прошлый элемент}

return true;}

return false;}

};

/\* Класс для работы с деком \*/

template<typename T>

class USEDECK {

public:

USEDECK() : Knot(NULL) {}

~USEDECK() {

while (DelRightElement());}

void menu() {

char choice;

system("cls");

cout << " {{>================================================<}}\n"

<< " || Список ||\n"

<< " {{>================================================<}}\n"

<< " || 1. Добавить элемент справа ||\n"

<< " || 2. Добавить элемент слева ||\n"

<< " || 3. Удалить элемент справа ||\n"

<< " || 4. Удалить элемент слева ||\n"

<< " || 5. Посмотреть связанный список ||\n"

<< " || 6. Поиск элемента ||\n"

<< " || 7. Выход ||\n"

<< " {{>================================================<}}\n"

<< "\n Выбранная опция: ";

(cin >> choice).get();

switch (choice) {

case 49: {

system("cls");

cout << "Введите новый элемент: ";

(cin >> Value).get();

AddRightElement(Value);

cout << "Добавлено. ";

system("pause");

menu();

break; }

case 50: {

system("cls");

cout << "Введите новый элемент: ";

(cin >> Value).get();

AddLeftElement(Value);

cout << "Добавлено. ";

system("pause");

menu();

break; }

case 51: {

system("cls");

if (DelRightElement())

cout << "Удалено. ";

else cout << "Нет списка. ";

system("pause");

menu();

break; }

case 52: {

system("cls");

if (DelLeftElement())

cout << "Удалено. ";

else cout << "Нет списка. ";

system("pause");

menu();

break; }

case 53: {

system("cls");

if (!PrintTheStucture())

cerr << "No structure";

cout << endl;

system("pause");

menu();

break; }

case 54: {

system("cls");

cout << "Введите элемент для поиска: ";

(cin >> Value).get();

ELEMENT<T>\* m = StructureSearch(Knot, Value);

cout << "\nНайденные элементы: " << endl;

if (m != NULL) {

for (int i = 0; i < (\_msize(m) / sizeof(ELEMENT<T>)); i++)

cout << setw(4) << m[i].inf; }

cout << endl;

system("pause");

menu();

break; }

case 55: {

break; }

default: menu(); }}

private:

/\* Структура элемента списка \*/

template<typename Ty>

struct ELEMENT {

Ty inf;

ELEMENT\* pLastElement;

ELEMENT\* pNewElement;

ELEMENT\* pOverLastElement; };

T Value;

ELEMENT<T>\* Knot;

public:

bool PrintTheStucture() {

if (Knot != NULL) {

ELEMENT<T>\* LastElement = Knot;

cout << "Elements:\n";

for (;;) {

cout << setw(4) << Knot->inf;

if (Knot->pLastElement == NULL) break;

Knot = Knot->pLastElement; }

Knot = LastElement;

return true; }

return false; }

template<typename ClassName, typename Type>

static ClassName\* StructureSearch(ClassName\* Element, Type value) {

if (Element != NULL) {

ClassName \*LastElement = Element;

int score = 0;

for (;;) {

if (Element->inf == value)

score++;

if (Element->pLastElement == NULL) break;

Element = Element->pLastElement;}

Element = LastElement;

ClassName \*m = new ClassName[score];

score = 0;

for (;;) {

if (Element->inf == value) {

m[score] = \*Element;

score++; }

if (Element->pLastElement == NULL) break;

Element = Element->pLastElement; }

return m; }

return NULL; }

template<typename Type>

inline void AddLeftElement(Type value) {

TRIPLEKNOTE::AddLeftElement(&Knot, value); }

inline bool DelRightElement(){

return TRIPLEKNOTE::DelRightElement(&Knot); }

bool DelLeftElement() {

return TRIPLEKNOTE::DelLeftElement(&Knot); }

template<typename Type>

inline void AddRightElement(Type value){

TRIPLEKNOTE::AddRightElement(&Knot, value); }};

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

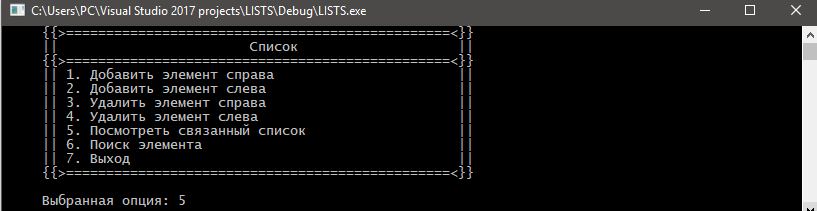
USEDECK<int> list;

list.menu();

return 0; }

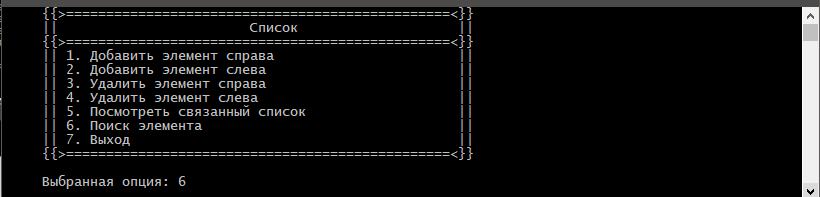
**Отладка**

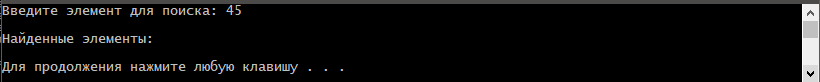
Просмотр элементов пустого трёхсвязного списка через меню выбора:



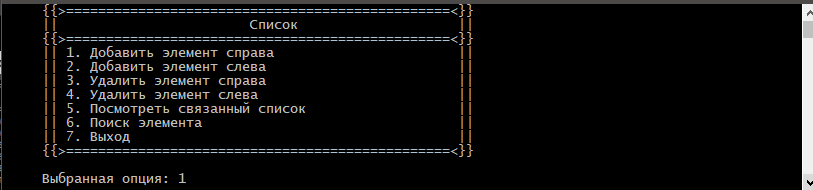


Поиск элемента в пустом трёхсвязном списке:



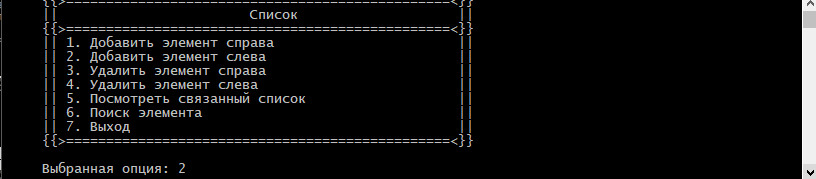


Добавление 2-х элементов справа и 2 элементов слева и просмотр через меню выбора:















Поиск существующего элемента списка:



Удаление всех элементов справа до тех пор пока не останется элементов в трёхсвязном списке:









