|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** ***ИУК «Информатика и управление»***

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №1**

**«Работа со списочными структурами»**

**ДИСЦИПЛИНА: «Типы и структуры данных»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-42Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Пчелинцева Н.И. )  (Подпись) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2022

**Цель:** формирование практических навыков создания алгоритмов обработки списочных структур данных.

**Задание:**

1. Изучить основные виды списочных структур.
2. Изучить организацию списочных структур.
3. Познакомиться с основными операциями для обработки списков.
4. Изучить типовые алгоритмы решения задач с использованием списков.
5. Реализовать основные алгоритмы обработки списочных структур данных (создание, удаление, поиск, добавление и удаление элемента), а также алгоритм согласно полученному варианту.

**Вариант 7**

Даны натуральное число n, действительные числа a1, a2, ..., an. Преобразовать последовательность a1, a2, ..., an, расположив вначале отрицательные члены, а затем – неотрицательные. При этом порядок отрицательных чисел сохраняется прежним, а порядок неотрицательных чисел изменяется на обратный (для решения задачи использовать линейный динамический список).

**Листинг:**

***List.h***

#pragma once

#include <vector>

#include "ElementAccessException.h"

namespace DTAS

{

class List

{

private:

struct Element

{

float data{};

Element\* next{};

};

Element\* \_head{};

Element\* \_currentElement{};

int \_currentPosition{};

public:

List();

List(std::vector<float> data);

std::vector<float> GetData();

void AppendHead(float data);

void AppendAfter(float data);

void AppendBefore(float data);

void Remove();

bool FindAndSet(float data);

bool IsEmpty();

void Clear();

void MoveForward();

float GetCurrentElement();

int GetCurrentPosition();

void ReturnBack();

};

}

***List.cpp***

#include "List.h"

namespace DTAS

{

List::List(): \_head(nullptr), \_currentElement(\_head), \_currentPosition(-1) {}

List::List(std::vector<float> data)

{

if (data.size() > 0)

{

\_head = new Element{ data[0], nullptr };

Element\* pointer = \_head;

for (unsigned short i = 1; i < data.size(); ++i)

{

pointer->next = new Element{ data[i], nullptr };

pointer = pointer->next;

}

\_currentPosition = 0;

}

else

{

\_head = nullptr;

\_currentPosition = -1;

}

\_currentElement = \_head;

}

std::vector<float> List::GetData()

{

std::vector<float> data{};

Element\* pointer = \_head;

while (pointer != nullptr)

{

data.push\_back(pointer->data);

pointer = pointer->next;

}

return data;

}

bool List::IsEmpty()

{

return \_head == nullptr;

}

void List::Clear()

{

while (\_head != nullptr)

{

Element\* pointer = \_head;

\_head = \_head->next;

delete pointer;

}

\_currentElement = \_head;

\_currentPosition = -1;

}

void List::AppendHead(float data)

{

if (IsEmpty())

{

\_head = new Element{ data, nullptr };

\_currentElement = \_head;

\_currentPosition = 0;

}

else

{

throw ElementAccessException("List isn't empty");

}

}

void List::AppendAfter(float data)

{

if (\_currentElement == nullptr)

{

throw ElementAccessException("Pointer error");

}

else

{

\_currentElement->next = new Element{ data, \_currentElement->next };

}

}

void List::AppendBefore(float data)

{

if (\_currentElement == nullptr)

{

throw ElementAccessException("Pointer error");

}

else

{

if (\_currentElement == \_head)

{

\_head = new Element{ data, \_currentElement };

}

else

{

Element\* pointer = \_head;

while (pointer->next != \_currentElement)

{

pointer = pointer->next;

}

pointer->next = new Element{ data, \_currentElement };

}

++\_currentPosition;

}

}

void List::Remove()

{

if (\_currentElement == nullptr)

{

throw ElementAccessException("Pointer error");

}

else

{

if (\_currentElement == \_head)

{

\_head = \_head->next;

delete \_currentElement;

\_currentElement = \_head;

}

else

{

Element\* pointer = \_head;

while (pointer->next != \_currentElement)

{

pointer = pointer->next;

}

\_currentElement = \_currentElement->next;

delete pointer->next;

pointer->next = \_currentElement;

}

if (\_currentElement == nullptr)

{

\_currentPosition = -1;

}

}

}

bool List::FindAndSet(float data)

{

Element\* pointer = \_currentElement;

int newPosition = \_currentPosition;

if (pointer != nullptr)

{

pointer = pointer->next;

++newPosition;

}

while (pointer != nullptr)

{

if (pointer->data == data)

{

\_currentElement = pointer;

\_currentPosition = newPosition;

return true;

}

else

{

pointer = pointer->next;

++newPosition;

}

}

pointer = \_head;

newPosition = 0;

while (pointer != \_currentElement)

{

if (pointer->data == data)

{

\_currentElement = pointer;

\_currentPosition = newPosition;

return true;

}

else

{

pointer = pointer->next;

++newPosition;

}

}

if (\_currentElement != nullptr && \_currentElement->data == data)

{

return true;

}

return false;

}

void List::MoveForward()

{

if (\_currentElement != nullptr)

{

\_currentElement = \_currentElement->next;

if (\_currentElement != nullptr)

{

++\_currentPosition;

}

else

{

\_currentPosition = -1;

}

}

}

float List::GetCurrentElement()

{

if (\_currentElement != nullptr)

{

return \_currentElement->data;

}

else

{

throw ElementAccessException("Pointer error");

}

}

int List::GetCurrentPosition()

{

return \_currentPosition;

}

void List::ReturnBack()

{

\_currentElement = \_head;

if (\_currentElement != nullptr)

{

\_currentPosition = 0;

}

else

{

\_currentPosition = -1;

}

}

}

***ElementAccessException.h***

#pragma once

#include <string>

namespace DTAS

{

class ElementAccessException

{

public:

ElementAccessException(std::string error);

std::string GetError();

private:

std::string \_error{};

};

}

***ElementAccessException.cpp***

#include "ElementAccessException.h"

namespace DTAS

{

ElementAccessException::ElementAccessException(std::string error) : \_error(error) {}

std::string ElementAccessException::GetError() { return \_error; }

}

***Menu.h***

#pragma once

#include "List.h"

namespace DTAS

{

class Menu

{

public:

Menu();

Menu(std::string input, std::string output, std::string logFile);

void Start();

private:

std::string \_output{};

std::string \_logFile{};

List \_list{};

void ShowMenu();

void ShowList();

void Add();

void Remove();

void Find();

void Clear();

void RunTask();

void MoveForward();

void ReturnBack();

};

}

***Menu.cpp***

#include "Menu.h"

#include <iostream>

#include <fstream>

namespace DTAS

{

Menu::Menu(): \_output("Output.txt"), \_logFile("Log.txt"), \_list() {}

Menu::Menu(std::string input, std::string output, std::string logFile) :

\_output(output), \_logFile(logFile)

{

std::ifstream inputData(input);

unsigned short number;

inputData >> number;

std::vector<float> data{};

for (unsigned short i{}; i < number; ++i)

{

float element;

inputData >> element;

data.push\_back(element);

}

inputData.close();

\_list = List(data);

}

void Menu::Start()

{

unsigned short command = 1;

while (command != 0)

{

system("cls");

ShowList();

std::cout << "\n";

ShowMenu();

std::cin >> command;

std::cout << "\n";

switch (command)

{

case 1:

Add();

break;

case 2:

Remove();

break;

case 3:

Find();

break;

case 4:

Clear();

break;

case 5:

RunTask();

break;

case 6:

MoveForward();

break;

case 7:

ReturnBack();

break;

}

}

}

void Menu::ShowMenu()

{

std::cout << "1. Add element\n";

std::cout << "2. Remove element\n";

std::cout << "3. Find element\n";

std::cout << "4. Clear list\n";

std::cout << "5. Run task\n";

std::cout << "6. Move forward\n";

std::cout << "7. Return back\n";

std::cout << "0. Exit\n";

std::cout << ">>> ";

}

void Menu::ShowList()

{

std::vector<float> data = \_list.GetData();

if (data.size() == 0)

{

std::cout << "List is empty\n";

}

else

{

int currentPosition = \_list.GetCurrentPosition();

for (unsigned short i{}; i < data.size(); ++i)

{

std::cout << i << ". ";

if (i == currentPosition)

{

std::cout << ">";

}

else

{

std::cout << " ";

}

std::cout << " " << data[i] << "\n";

}

}

}

void Menu::Add()

{

if (\_list.IsEmpty())

{

std::cout << "Number: ";

float number;

std::cin >> number;

\_list.AppendHead(number);

}

else

{

unsigned short mode;

std::cout << "0. Add before\n";

std::cout << "1. Add after\n";

std::cout << ">>> ";

std::cin >> mode;

if (mode != 0 && mode != 1)

{

std::cout << "Input error\n";

return;

}

std::cout << "\nNumber: ";

float number;

std::cin >> number;

try

{

if (mode == 0)

{

\_list.AppendBefore(number);

}

else if (mode == 1)

{

\_list.AppendAfter(number);

}

}

catch (ElementAccessException& error)

{

std::cout << error.GetError() << "\n";

std::ofstream errorData(\_logFile, std::ios\_base::app);

errorData << "Adding error\n";

errorData.close();

system("pause");

}

}

}

void Menu::Remove()

{

try

{

\_list.Remove();

}

catch (ElementAccessException& error)

{

std::cout << error.GetError() << "\n";

std::ofstream errorData(\_logFile, std::ios\_base::app);

errorData << "Removing error\n";

errorData.close();

system("pause");

}

}

void Menu::Find()

{

std::cout << "Number: ";

float number;

std::cin >> number;

if (!\_list.FindAndSet(number))

{

std::cout << "Number wasn't found\n";

system("pause");

}

}

void Menu::Clear()

{

\_list.Clear();

}

void Menu::RunTask()

{

if (!\_list.IsEmpty())

{

ReturnBack();

List newList{};

while (\_list.GetCurrentPosition() != -1)

{

float number = \_list.GetCurrentElement();

if (number < 0)

{

if (newList.IsEmpty())

{

newList.AppendHead(number);

}

else

{

newList.AppendAfter(number);

newList.MoveForward();

}

}

\_list.MoveForward();

}

ReturnBack();

while (\_list.GetCurrentPosition() != -1)

{

float number = \_list.GetCurrentElement();

if (number >= 0)

{

if (newList.IsEmpty())

{

newList.AppendHead(number);

}

else

{

newList.AppendAfter(number);

}

}

\_list.MoveForward();

}

std::vector<float> data = newList.GetData();

std::ofstream outputData(\_output);

if (data.size() == 0)

{

std::cout << "List is empty\n";

}

else

{

for (unsigned short i{}; i < data.size(); ++i)

{

std::cout << i << ". " << data[i] << "\n";

outputData << data[i] << " ";

}

}

outputData.close();

ReturnBack();

std::cout << "\n";

system("pause");

}

}

void Menu::MoveForward()

{

\_list.MoveForward();

}

void Menu::ReturnBack()

{

\_list.ReturnBack();

}

}

***Main.cpp***

#include "Menu.h"

#include <iostream>

int main(int argc, char\* argv[])

{

if (argc == 1)

{

DTAS::Menu menu{};

menu.Start();

}

if (argc == 4)

{

DTAS::Menu menu{ argv[1], argv[2], argv[3] };

menu.Start();

}

else

{

std::cout << "Too few arguments\n";

}

return 0;

}

**Результат:**



Рис. 1. Запуск через консоль

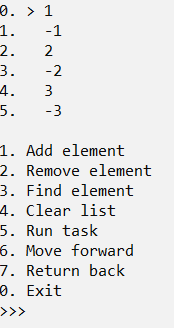


Рис. 2. Главное меню

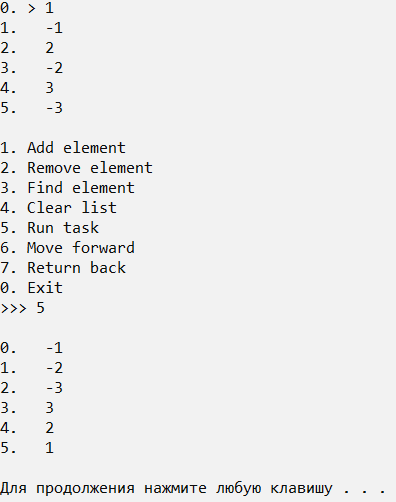


Рис. 3. Решение задачи

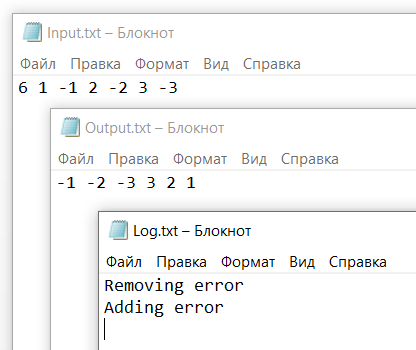


Рис. 4. Файлы

**Вывод:** в ходе выполнения лабораторной работы были получены навыки написания и дальнейшего использования собственного однонаправленного списка, реализации вызова программы через консоль с использованием нескольких параметров.