|  |  |
| --- | --- |
| **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. ИУК4-32Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_Зудин Д.В.\_\_\_\_\_)  (Подпись) (Ф.И.О.) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_Пчелинцева Н.И.\_\_\_)  (Подпись) (Ф.И.О.) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |
| Калуга, 2022 г.  **Цель**: формирование практических навыков создания алгоритмов обработки списочных структур данных.  **Задачи**:  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 №24**  **Формулировка задания**  1. Разработать консольное приложение, написанное с помощью объектно-ориентированной технологии. Индивидуальное задание предусмотрено вариантом, который назначает преподаватель.  2. Приложение необходимо запускать для демонстрации из командной строки с указанием названий приложения и трех файлов:  − все входные данные (например, последовательности чисел, коэффициенты многочленов и т.д.) считать из первого файла;  − все выходные данные записать во второй файл;  − все возникшие ошибки записать в третий файл – файл ошибок.  3. Все основные сущности приложения представить в виде отдельных классов.  4. Необходимо предусмотреть пользовательское меню, содержащее набор команд всех основных операций для работы со списком, а также команду для запуска индивидуального задания.  5. В приложении также должны быть учтены все критические ситуации, обработанные с помощью класса исключений.  **Индивидуальное задание**  Дана последовательность символов, оканчивающаяся точкой. Удалить все символы, у которых равные соседи (первый и последний символы считать соседями) (для решения задачи использовать линейный двусвязный динамический список).  **Листинг файла DoubleList.h**  #pragma once  #include <iostream>  class DoubleList  {  public:  struct Node  {  int data;  Node\* next;  Node\* prev;  };  DoubleList();  ~DoubleList();  bool IsEmpty() const;  void PrintDoubleList() const;  void PushBack(int data);  void PushFront(int data);  void Insert(size\_t index, int data);  void PopBack();  void PopFront();  void Remove(size\_t index);  void Clear();  int FindElement(int data) const;  int RFindElement(int data) const;  size\_t getSize() const;  Node\* getHead() const;  Node\* getTail() const;  friend std::ostream& operator<<(std::ostream& out, const DoubleList& dl);  private:  Node\* head;  Node\* tail;  size\_t size;  };  **Листинг файла DoubleList.cpp**  #include "DoubleList.h"  DoubleList::DoubleList()  {  head = nullptr;  tail = nullptr;  size = 0;  }  DoubleList::~DoubleList()  {  if (IsEmpty())  {  return;  }  Node\* curr = head->next;  while (curr)  {  delete curr->prev;  curr = curr->next;  }  delete tail;  }  bool DoubleList::IsEmpty() const  {  return !size;  }  void DoubleList::PrintDoubleList() const  {  std::cout << \*this;  }  void DoubleList::PushBack(int data)  {  Node\* temp = new Node;  temp->data = data;  if (IsEmpty())  {  temp->prev = nullptr;  head = temp;  }  else  {  tail->next = temp;  temp->prev = tail;  }  temp->next = nullptr;  tail = temp;  size++;  }  void DoubleList::PushFront(int data)  {  Node\* temp = new Node;  temp->data = data;  if (IsEmpty())  {  temp->next = nullptr;  tail = temp;  }  else  {  head->prev = temp;  temp->next = head;  }  temp->prev = nullptr;  head = temp;  size++;  }  void DoubleList::Insert(size\_t index, int data)  {  if (index >= 0 && index <= size)  {  if (index == 0)  {  PushFront(data);  return;  }  if (index == size)  {  PushBack(data);  return;  }  Node\* currLeft = head;  for (int i = 0; i < index - 1; i++)  {  currLeft = currLeft->next;  }  Node\* currRight = currLeft->next;  Node\* temp = new Node;  temp->data = data;  temp->prev = currLeft;  temp->next = currRight;  currLeft->next = temp;  currRight->prev = temp;  size++;  }  }  void DoubleList::PopBack()  {  if (IsEmpty())  {  return;  }  if (size == 1)  {  delete head;  head = nullptr;  tail = nullptr;  size--;  return;  }  tail = tail->prev;  delete tail->next;  tail->next = nullptr;  size--;  }  void DoubleList::PopFront()  {  if (IsEmpty())  {  return;  }  if (size == 1)  {  delete head;  head = nullptr;  tail = nullptr;  size--;  return;  }  head = head->next;  delete head->prev;  head->prev = nullptr;  size--;  }  void DoubleList::Remove(size\_t index)  {  if (index >= 0 && index <= size)  {  if (IsEmpty())  {  return;  }  if (index == 0)  {  PopFront();  return;  }  if (index == size - 1)  {  PopBack();  return;  }  Node\* currLeft = head;  for (int i = 0; i < index - 1; i++)  {  currLeft = currLeft->next;  }  Node\* currRight = currLeft->next->next;  currLeft->next = currRight;  delete currRight->prev;  currRight->prev = currLeft;  size--;  }  }  void DoubleList::Clear()  {  this->~DoubleList();  head = nullptr;  tail = nullptr;  size = 0;  }  int DoubleList::FindElement(int data) const  {  size\_t index = 0;  Node\* curr = head;  bool find = false;  while (curr)  {  if (curr->data == data)  {  find = true;  break;  }  curr = curr->next;  index++;  }  return (find ? index : -1);  }  int DoubleList::RFindElement(int data) const  {  size\_t index = size - 1;  Node\* curr = tail;  bool find = false;  while (curr)  {  if (curr->data == data)  {  find = true;  break;  }  curr = curr->prev;  index--;  }  return (find ? index : -1);  }  size\_t DoubleList::getSize() const  {  return size;  }  DoubleList::Node\* DoubleList::getHead() const  {  return head;  }  DoubleList::Node\* DoubleList::getTail() const  {  return tail;  }  std::ostream& operator<<(std::ostream& out, const DoubleList& dl)  {  DoubleList::Node\* curr = dl.getHead();  out << "[";  while (curr)  {  out << curr->data << (curr->next ? ", " : "");  curr = curr->next;  }  out << "] \nsize: " << dl.getSize() << "\n";  return out;  }  **Листинг файла FileLogging.h**  #pragma once  #ifdef \_MSC\_VER  #define \_CRT\_SECURE\_NO\_WARNINGS  #endif  #include <string>  #include <fstream>  #include <ctime>  #include <iostream>  class FileLogging  {  public:  FileLogging(std::string fileName);  void Logging(std::string message);  void PrintFile();  private:  std::string getTime();  std::string fileName;  };  **Листинг файла FileLogging.cpp**  #include "FileLogging.h"  FileLogging::FileLogging(std::string fileName)  {  this->fileName = fileName;  }  void FileLogging::Logging(std::string message)  {  std::ofstream fout(fileName, std::ios::out | std::ios::app);  if (fout.is\_open())  {  fout << "[" << getTime() << "]" << message;  }  fout.close();  }  std::string FileLogging::getTime()  {  time\_t seconds = time(NULL);  tm\* timeinfo = localtime(&seconds);  std::string currTime = asctime(timeinfo);  currTime.pop\_back();  return currTime;  }  void FileLogging::PrintFile()  {  std::ifstream fin(fileName, std::ios::in);  std::string temp;  std::cout << fileName << ":\n";  if (fin.is\_open())  {  while (std::getline(fin, temp))  {  std::cout << temp << std::endl;  }  }  }  **Листинг файла DTaS\_Lab1.cpp**  #include "DoubleList.h"  #include "FileLogging.h"  #include <sstream>  #include <vector>  enum Points  {  CREATE\_LIST = 1,  PRINT\_LIST,  INSERT\_IN\_LIST,  DELETE\_FROM\_LIST,  CLEAR\_LIST,  FIND\_ELEMENT,  CHECK\_EMPTY,  GET\_LENGTH,  INDIVIDUAL\_TASK,  DELETE\_LIST,  OPEN\_ERROR\_LOG,  OPEN\_OUTPUT\_LOG,  INPUT\_DATA\_FROM\_FILE,  EXIT  };  bool IsDigit(char c);  int Input(std::string message, int min, int max, FileLogging& f1);  void InputDataFromFile(DoubleList\* dl);  void PrintList(DoubleList\* dl, FileLogging& f1);  void InsertInList(DoubleList\* dl, FileLogging& f1);  void DeleteFromList(DoubleList\* dl, FileLogging& f1);  void ClearList(DoubleList\* dl, FileLogging& f1);  void FindElement(DoubleList\* dl, FileLogging& f1);  void CheckEmpty(DoubleList\* dl, FileLogging& f1);  void GetLength(DoubleList\* dl, FileLogging& f1);  void IndividualTask();  int main()  {  setlocale(LC\_ALL, "Russian");  DoubleList\* dl = nullptr;  bool exit = false;  FileLogging errorLog("error\_log.txt");  FileLogging outputLog("output\_log.txt");  outputLog.Logging("Program is launched.\n");  while (!exit)  {  if (dl)  {  std::stringstream ss;  ss << \*dl;  outputLog.Logging(ss.str());  }  else  {  outputLog.Logging("List does not exist.\n");  }  std::cout <<  "--------------------------Меню---------------------------\n" <<  " 1. Создать список\n" <<  " 2. Вывести список\n" <<  " 3. Вставить элемент в список\n" <<  " 4. Удалить элемент из списка\n" <<  " 5. Очистить список\n" <<  " 6. Найти элемент в списке\n" <<  " 7. Проверить список на пустоту\n" <<  " 8. Узнать длину списка\n" <<  " 9. Индивидуальное задание\n" <<  "10. Удалить список\n" <<  "11. Открыть error\_log.txt\n" <<  "12. Открыть output\_log.txt\n" <<  "13. Считать данные из input.txt\n" <<  "14. Выход\n" <<  "----------------------------------------------------------\n";  int choice = Input("Выбрать: ", 1, 14, errorLog);  system("cls");  switch (choice)  {  case CREATE\_LIST:  if (!dl)  {  dl = new DoubleList();  std::cout << "Список успешно создан!\n";  }  else  {  std::cout << "Список уже создан!\n";  errorLog.Logging("List already created: attempt to create a list.\n");  }  break;  case PRINT\_LIST:  PrintList(dl, errorLog);  break;  case INSERT\_IN\_LIST:  InsertInList(dl, errorLog);  break;  case DELETE\_FROM\_LIST:  DeleteFromList(dl, errorLog);  break;  case CLEAR\_LIST:  ClearList(dl, errorLog);  break;    case FIND\_ELEMENT:  FindElement(dl, errorLog);  break;  case CHECK\_EMPTY:  CheckEmpty(dl, errorLog);  break;  case GET\_LENGTH:  GetLength(dl, errorLog);  break;  case INDIVIDUAL\_TASK:  IndividualTask();  break;  case DELETE\_LIST:  if (!dl)  {  std::cout << "Список еще не создан!\n";  errorLog.Logging("List does not exist: attempt to delete list.\n");  }  else  {  delete dl;  dl = nullptr;  std::cout << "Список успешно удален!\n";  }  break;  case OPEN\_ERROR\_LOG:  errorLog.PrintFile();  break;  case OPEN\_OUTPUT\_LOG:  outputLog.PrintFile();  break;  case INPUT\_DATA\_FROM\_FILE:  if (!dl)  {  dl = new DoubleList();  }  InputDataFromFile(dl);  break;  case EXIT:  exit = true;  break;  }  std::cout << "\n";  system("pause");  system("cls");  }  if (dl)  {  delete dl;  }  return 0;  }  bool IsDigit(char c)  {  return '0' <= c && c <= '9';  }  int Input(std::string message, int min, int max, FileLogging& f1)  {  int number = 0;  bool correct = false;  while (!correct)  {  std::cout << message;  std::string input = "";  std::cin >> input;  correct = (input[0] == '-' || IsDigit(input[0]));  for (size\_t i = 1; i < input.size(); i++)  {  correct = IsDigit(input[i]);  if (!correct)  {  break;  }  }  if (!correct)  {  std::cout << "Неккоректная запись числа!\n";  f1.Logging("Incorrect number entry.\n");  }  if (correct && input.size() > std::to\_string(INT\_MAX).size() - 1)  {  correct = false;  std::cout << "Введенное число выходит из допустимого диапазона!\n";  f1.Logging("The entered number out of range.\n");  }  if (correct)  {  number = stoi(input);  if (min > number || max < number)  {  correct = false;  std::cout << "Введенное число выходит из допустимого диапазона!\n";  f1.Logging("The entered number out of range.\n");  }  }  }  return number;  }  void InputDataFromFile(DoubleList\* dl)  {  dl->Clear();  std::fstream fin("input.txt", std::ios::in);  if (fin.is\_open())  {  int data;  while (fin >> data)  {  dl->PushBack(data);  }  }  std::cout << "Данные успешно считались!\n";  fin.close();  }  void PrintList(DoubleList\* dl, FileLogging& fl)  {  if (dl)  {  dl->PrintDoubleList();  }  else  {  std::cout << "Список еще не создан!\n";  fl.Logging("List does not exist: attempt to print list.\n");  }  }  void InsertInList(DoubleList\* dl, FileLogging& fl)  {  if (dl)  {  std::cout <<  "--------------------------Вставка---------------------------\n" <<  " 1. Вставить элемент в начало списка\n" <<  " 2. Вставить элемент в конец списка\n" <<  " 3. Вставить элемент в список по индексу\n" <<  " 4. Выйти в главное меню\n" <<  "-------------------------------------------------------------\n";  int subchoice = Input("Выбрать: ", 1, 4, fl);  int data = 0;  int index = 0;  switch (subchoice)  {  case 1:  data = Input("Введите число для вставки: ", INT\_MIN, INT\_MAX, fl);  dl->PushFront(data);  std::cout << "Число " << data << " успешно добавлено в начало списка!\n";  break;  case 2:  data = Input("Введите число для вставки: ", INT\_MIN, INT\_MAX, fl);  dl->PushBack(data);  std::cout << "Число " << data << " успешно добавлено в конец списка!\n";  break;  case 3:  data = Input("Введите число для вставки: ", INT\_MIN, INT\_MAX, fl);  index = Input("Введите индекс элемента для вставки: ", 0, dl->getSize(), fl);  dl->Insert(index, data);  std::cout << "Число " << data << " успешно добавлено в позицию с номером " << index << "списка!\n";  break;  case 4:  break;  }  }  else  {  std::cout << "Список еще не создан!\n";  fl.Logging("List does not exist: attempt to add a new element to the list.\n");  }  }  void DeleteFromList(DoubleList\* dl, FileLogging& fl)  {  if (dl)  {  if (dl->IsEmpty())  {  std::cout << "Нельзя ничего удалить из пустого списка!\n";  fl.Logging("List is empty: attempt to remove an element from a list.\n");  }  else  {  std::cout <<  "--------------------------Удаление---------------------------\n" <<  " 1. Удалить первый элемент списка\n" <<  " 2. Удалить последний элемент списка\n" <<  " 3. Удалить элемент из списка по индексу\n" <<  " 4. Выйти в главное меню\n" <<  "-------------------------------------------------------------\n";  int subchoice = Input("Выбрать: ", 1, 4, fl);  int index = 0;  switch (subchoice)  {  case 1:  dl->PopFront();  std::cout << "Первый элемент успешно удалён!\n";  break;  case 2:  dl->PopBack();  std::cout << "Последний элемент успешно удалён!\n";  break;  case 3:  index = Input("Введите индекс элемента для удаления: ", 0, dl->getSize() - 1, fl);  dl->Remove(index);  std::cout << "Элемент с индексом " << index << " успешно удалён из списка!\n";  break;  case 4:  break;  }  }  }  else  {  std::cout << "Список еще не создан!\n";  fl.Logging("List does not exist: attempt to remove an element from a list.\n");  }  }  void ClearList(DoubleList\* dl, FileLogging& fl)  {  if (dl)  {  dl->Clear();  std::cout << "Список успешно очищен!\n";  }  else  {  std::cout << "Список еще не создан!\n";  fl.Logging("List does not exist: trying to clear the list.\n");  }  }  void FindElement(DoubleList\* dl, FileLogging& fl)  {  if (dl)  {  std::cout <<  "--------------------------Поиск------------------------------\n" <<  " 1. Первое вхождение элемента слева\n" <<  " 2. Первое вхождение элемента справа\n" <<  " 3. Выйти в главное меню\n" <<  "-------------------------------------------------------------\n";  int subchoice = Input("Выбрать: ", 1, 3, fl);  int data = 0;  switch (subchoice)  {  case 1:  data = Input("Введите число для поиска: ", INT\_MIN, INT\_MAX, fl);  std::cout << "Индекс этого элемента: " << dl->FindElement(data) << "\n";  break;  case 2:  data = Input("Введите число для поиска: ", INT\_MIN, INT\_MAX, fl);  std::cout << "Индекс этого элемента: " << dl->RFindElement(data) << "\n";  break;  case 3:  break;  }  }  else  {  std::cout << "Список еще не создан!\n";  fl.Logging("List does not exist: attempt to find an element in a list.\n");  }  }  void CheckEmpty(DoubleList\* dl, FileLogging& fl)  {  if (dl)  {  std::cout << "Список пуст: " << (dl->IsEmpty() ? "да" : "нет") << "\n";  }  else  {  std::cout << "Список еще не создан!\n";  fl.Logging("List does not exist: attempt to check if the list is empty.\n");  }  }  void GetLength(DoubleList\* dl, FileLogging& fl)  {  if (dl)  {  std::cout << "Длина списка: " << dl->getSize() << "\n";  }  else  {  std::cout << "Список еще не создан!\n";  fl.Logging("List does not exist: attempt to get the lenght of the list.\n");  }  }  void IndividualTask()  {  std::string data;  std::cout << "Введите строку: ";  std::cin >> data;  DoubleList\* tdl = new DoubleList();  for (auto i : data)  {  tdl->PushBack((int)i);  }  DoubleList::Node\* curr = tdl->getHead()->next;  std::vector<DoubleList::Node\*> elements;  for (size\_t i = 1; i < tdl->getSize() - 2; i++)  {  if (curr->prev->data == curr->next->data)  {  curr->data = INT\_MAX;  }  curr = curr->next;  }  if (curr->prev->data == tdl->getHead()->data)  {  tdl->getTail()->prev->data = INT\_MAX;  }  bool exist = true;  while (exist)  {  exist = false;  curr = tdl->getHead();  for (size\_t i = 0; i < tdl->getSize() - 1; i++)  {  if (curr->data == INT\_MAX)  {  tdl->Remove(i);  exist = true;  break;  }  curr = curr->next;  }  }  curr = tdl->getHead();  std::cout << "Результат: ";  while (curr)  {  std::cout << static\_cast<char>(curr->data);  curr = curr->next;  }  std::cout << "\n";  delete tdl;  }  **Результат выполнения программы для задания**                              **Выводы:**  В ходе работы были сформированы практические навыки создания алгоритмов обработки списочных структур данных. | | |