|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 5** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Однонаправленный динамический список»** | |
|  | |
| Выполнил студент группы ИКБО-09-22 | Гришин А. В. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2023

# **Цель работы**

Получить знания и практические навыки управления динамическим однонаправленным списком в языке программирования С++.

1. **Постановка задачи**

Реализуйте программу решения задачи варианта по использованию линейного однонаправленного списка.

1. Разработать функцию для создания исходного списка, его вывода и добавления узла.
2. Информационная часть узла списка определена вариантом.
3. Разработать функции дополнительного задания варианта.
4. В основной программе выполнить тестирование каждой функции, описанной в задании.
5. Составить отчет по выполненному заданию. В отчет включить ответы на вопросы к практической работе.

Персональный вариант – 7.

Таблица 1. Варианты заданий

|  |  |
| --- | --- |
| Тип данных  int | Дан линейный однонаправленный список L   1. Разработать функцию, которая проверяет, есть ли в списке L два одинаковых элемента. 2. Разработать функцию, которая удаляет из списка L максимальное значение. 3. Разработать функцию, которая вставляет в список L новое значение перед каждым узлом в четной позиции. |

# **3. Решение**

* 1. **Краткая теория по теме**

Однонаправленный список – это структура данных, представляющая собой последовательность элементов, в каждом из которых хранится значение и указатель на следующий элемент списка. В последнем элементе указатель на следующий элемент равен nullptr.

|  |
| --- |
| struct Node  {  int value;  Node\* Next = nullptr;  }; |

Элемент, на который нет указателя, является первым (головным) элементом списка. Таким образом, если в массиве положение элементов определяется индексами, то в связном списке - указателями на следующий и (или) на предыдущий элемент.

Связные списки могут различаться. Есть односвязные списки, в которых каждый узел хранит указатель только на следующий узел. Есть двусвязные списки: в них каждый элемент хранит ссылку как на следующий элемент, так и на предыдущий. Есть кольцевые замкнутые списки.

**3.2. Ответ на вопрос к практической работе - дайте определение структуре данных стек.**

Стек - это абстрактный тип данных, представляющий собой коллекцию элементов, упорядоченных по принципу "последним пришел - первым вышел" (Last-In-First-Out, LIFO). Стек может быть реализован как набор элементов, расположенных на линейной структуре данных, где доступ к элементам осуществляется только через один конец структуры, называемый "вершиной стека". Операции добавления нового элемента в стек (push) и извлечения последнего элемента из стека (pop) выполняются только на вершине стека. Кроме того, для работы со стеком могут быть определены другие операции, такие как проверка на пустоту стека (empty), получение значения элемента на вершине стека без его удаления (top) и определение размера стека (size). Стек используется во многих алгоритмах и программах для решения различных задач, таких как обход деревьев, вычисление арифметических выражений, работа с рекурсивными функциями и многих других.

* 1. **. Дополнительные функции.**

Для упрощения работы над реализацией программы и повышения читаемости кода введем дополнительные функции: ListOutput - для вывода списка на экран, AddNewNode – для добавления нового узла в список.

|  |
| --- |
| void ListOutput(auto NodeList) {  if (NodeList->Next != nullptr) {  cout << NodeList->value << " -> ";  ListOutput(NodeList->Next);  }  else {  cout << NodeList->value;  }  }  void AddNewNode(auto& ListRoot, int newValue) {  if (ListRoot == nullptr) {  ListRoot = new Node{ newValue };  }  else {  Node\* current = ListRoot;  while (current->Next != nullptr) current = current->Next;  current->Next = new Node{ newValue };  }  } |

В функции ListOutput производиться вывод всех узлов списка со специальным знаком "стрелки" для лучшей читаемости. Используя рекурсию, вывод узлов списка требует меньше строк кода и времени для выполнения операции.

Функция AddNewNode принимает два параметра – корневой узел и значение (newValue) нового узла. Данная функция также используется для инициализации списка, так как если корневой узел списка отсутствует, то выполняется его создание и выход из функции. Если же таковой узел присутствует, то функция идет до последнего узла и присваивает указателю на следующий узел структуру со значением входного параметра newValue.

* 1. **Выполнение первого задания - разработать функцию, которая проверяет, есть ли в списке L два одинаковых элемента.**

Для выполнения данного задания создается пустой массив A и, вместе с указателем на корневой узел списка, передаётся в функцию FirstTask, отвечающую за выполнение первого задания.

|  |
| --- |
| bool FirstTask(auto& A, Node\* ListNode) {  while (ListNode != nullptr) {  Node\* curr = ListNode->Next;  while (curr != nullptr) {  if (ListNode->value == curr->value) {  return true;  }  curr = curr->Next;  }  ListNode = ListNode->Next;  }  return false;  } |

Данная функция выполняет поиск повторяющихся значений в связном списке, представленном указателем на его первый узел ListNode. Она проверяет каждую пару узлов списка, начиная с первого и до последнего, сравнивая значение текущего узла ListNode с каждым узлом, следующим за ним. Если находится пара узлов, в которых значения совпадают, функция возвращает true, что означает, что в списке есть повторяющиеся значения. Если функция перебрала все возможные пары и не нашла повторяющихся значений, она возвращает false.

* 1. **Выполнение второго задания - разработать функцию, которая удаляет из списка L максимальное значение.**

Для выполнения данного задания нам необходимо удалить все узлы списка из памяти, для его полного удаления. Так вызывается функция SecondTask с аргументом корня списка.

|  |
| --- |
| void SecondTask(Node\*& NodeList) {  if (NodeList == nullptr) {  return;  }  Node\* maxNode = NodeList;  Node\* prevMaxNode = nullptr;  Node\* curr = NodeList->Next;  while (curr != nullptr) {  if (curr->value > maxNode->value) {  maxNode = curr;  prevMaxNode = NodeList;  }  else if (curr->value == maxNode->value) {  prevMaxNode = maxNode;  }  curr = curr->Next;  }  if (maxNode == NodeList) {  NodeList = NodeList->Next;  }  else if (prevMaxNode && prevMaxNode->Next != maxNode) {  prevMaxNode = prevMaxNode->Next;  }  if (prevMaxNode) {  prevMaxNode->Next = maxNode->Next;  }  else {  NodeList = maxNode->Next;  }  delete maxNode;  } |

Данная функция удаляет узел с максимальным значением из связного списка, на который указывает указатель NodeList. Она проходит по списку, начиная со второго узла, и ищет узел с максимальным значением. В процессе поиска функция сохраняет указатель на узел с максимальным значением и указатель на предыдущий узел. Если максимальное значение находится в первом узле списка, то он удаляется путем изменения указателя NodeList на указатель на следующий узел. Если максимальное значение находится в узле списка, отличном от первого, то указатель prevMaxNode устанавливается на следующий узел после maxNode. Затем узел maxNode удаляется из списка путем изменения указателя на следующий узел узла, на который указывает prevMaxNode. Если maxNode был первым узлом, то изменяется указатель NodeList. В конце функция освобождает память, выделенную под узел maxNode.

* 1. **Выполнение третьего задания - разработать функцию, которая вставляет в список L новое значение перед каждым узлом в четной позиции.**

Для выполнения последнего задания вызывается функция ThirdExercise, в которую передается указатель на корневой узел и значение newValue, которое необходимо вставить согласно условию.

|  |
| --- |
| void ThirdTask(Node\*& L, int newValue) {  Node\* current = L;  int count = 1;  while (current != nullptr) {  if (count % 2 == 0) {  Node\* newNode = new Node;  newNode->value = newValue;  newNode->Next = current;  Node\* prev = L;  while (prev->Next != current) {  prev = prev->Next;  }  prev->Next = newNode;  }  current = current->Next;  count++;  }  } |

Данная функция вставляет новый узел со значением newValue после каждого второго узла в связном списке, на который указывает указатель L. Она проходит по списку, начиная с первого узла, и для каждого второго узла создает новый узел с заданным значением newValue. Затем новый узел вставляется между текущим узлом и предыдущим узлом, используя указатели на эти узлы. Для поиска предыдущего узла функция проходит по списку от начала до текущего узла, сохраняя указатель на предыдущий узел в переменной prev. В конце функции список будет содержать новые узлы со значениями newValue, добавленные после каждого второго узла в исходном списке.

* 1. **Описание работы программы и её интерфейса**

В самом начале программы выводится список L (изначально пустой) и меню выбора задания (рис. 1).

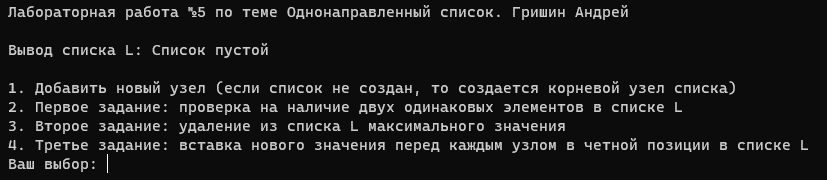


Рисунок 1. Интерфейс работы программы

В меню также присутствует отдельный пункт для добавления числа в список. Так для демонстрации работы программы добавим пару тестовых значений в список (рис. 2).

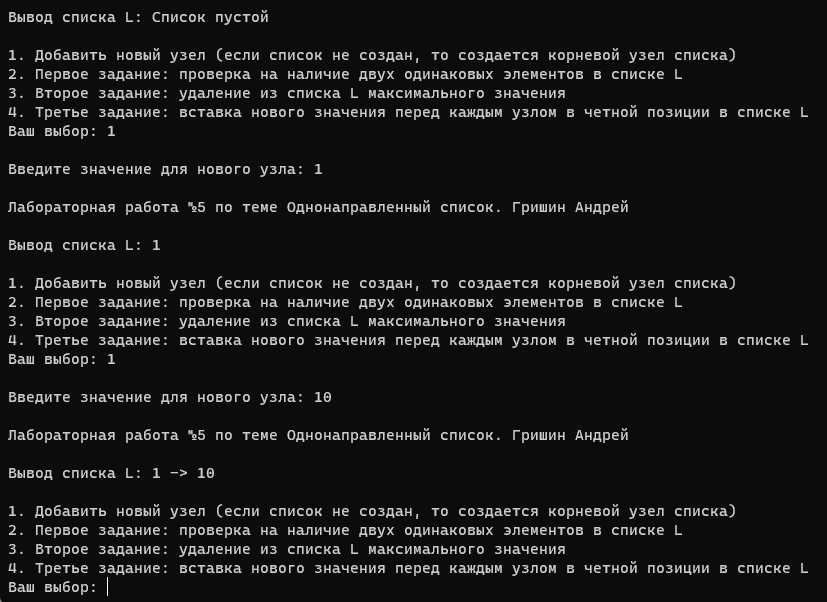


Рисунок 2. Демонстрация добавления нового узла в список

1. **Тестирование**

Протестируем работу программы для всех заданий. Для тестирования первого задания возьмем список с некоторыми одинаковыми элементами (рис. 3) и список с полностью уникальными (рис. 4).

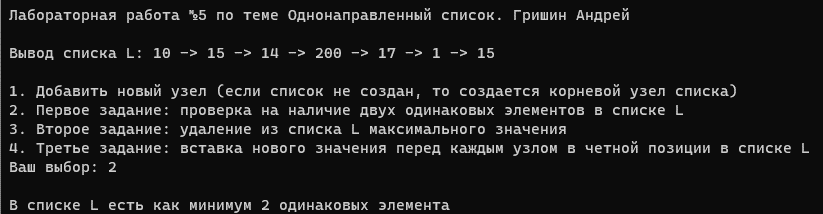


Рисунок 3. Тестирование работы первого задания со списком с некоторыми одинаковыми элементами

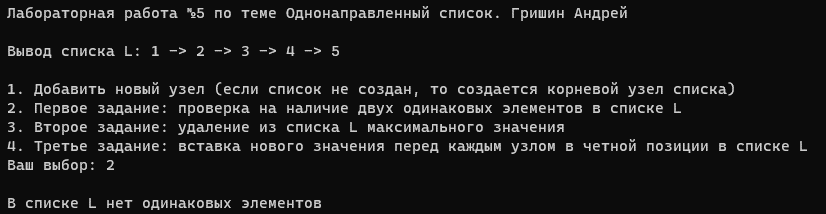


Рисунок 4. Тестирование работы первого задания со списком с полностью уникальными элементами

Тестирование первого задания завершено успешно. Протестируем работу второго задания на заполненном списке L (рис. 5) и на пустом списке (рис. 6).

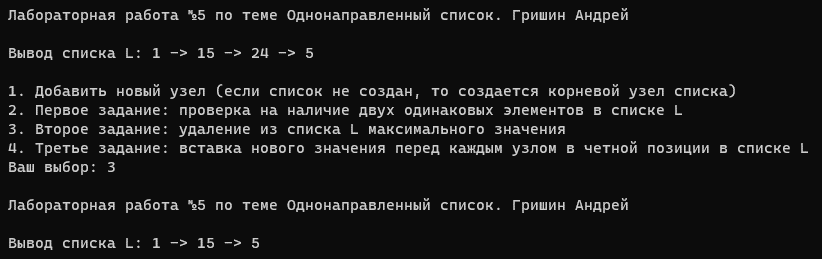


Рисунок 5. Тестирование работы второго задания с полным списком

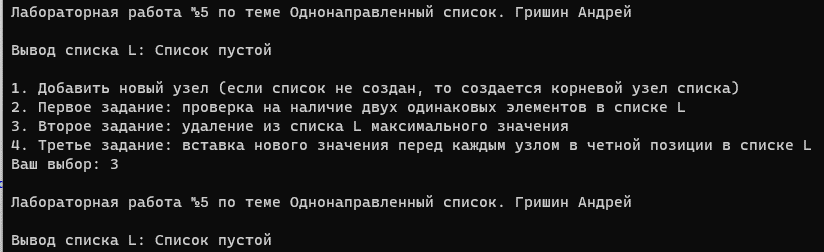


Рисунок 6. Тестирование работы второго задания с пустым списком

Тестирование второго задания завершено успешно. Протестируем работу третьего задания при заполненном списке (рис. 7) и пустом списке (рис. 8).

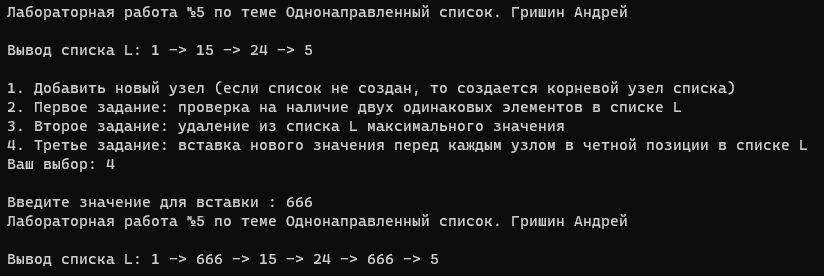


Рисунок 7. Тестирование работы третьего задания с заполненным списком

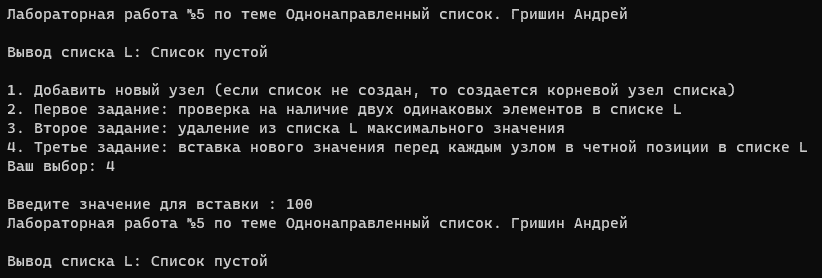


Рисунок 8. Тестирование работы третьего задания с пустым списком

Тестирование третьей операции завершено успешно. В данном тестировании мы показали, как работает третье задание при заполненном списке и пустом (в этом случае, список так и остаётся пустым, т.к. отсутствуют чётные элементы).

1. **Вывод**

В результате выполнения работы я получил знания и практические навыки управления динамическим однонаправленным списком в языке программирования С++.

1. **Исходный код программы**

Исходный код файла main.cpp

|  |
| --- |
| #include <iostream>  #include "NodeList.h"  using namespace std;  int main()  {  setlocale(LC\_ALL, "Rus");  Node\* L\_root = nullptr;  Node\* A[10];  for (int i = 0; i < 10; i++)  A[i] = nullptr;  int input;  int choice;  while (true)  {  cout << "Лабораторная работа №5 по теме Однонаправленный список. Гришин Андрей" << endl << endl;  cout << "Вывод списка L: ";  if (L\_root == nullptr)  {  cout << "Список пустой";  }  else  {  ListOutput(L\_root);  }  cout << endl;  cout << endl;  cout << "1. Добавить новый узел (если список не создан, то создается корневой узел списка)" << endl;  cout << "2. Первое задание: проверка на наличие двух одинаковых элементов в списке L" << endl;  cout << "3. Второе задание: удаление из списка L максимального значения" << endl;  cout << "4. Третье задание: вставка нового значения перед каждым узлом в четной позиции в списке L" << endl;  cout << "Ваш выбор: "; cin >> choice;  cout << endl;  switch (choice)  {  case 1:  {  cout << "Введите значение для нового узла: ";  cin >> input;  AddNewNode(L\_root, input);  cout << endl;  break;  }  case 2:  {  for (int i = 0; i < 10; i++)  A[i] = nullptr;  if (FirstTask(A, L\_root)) {  cout << "В списке L есть как минимум 2 одинаковых элемента";  }  else {  cout << "В списке L нет одинаковых элементов";  }  cout << endl;  break;  }  case 3:  {  SecondTask(L\_root);  break;  }  case 4:  {  int newValue;  cout << "Введите значение для вставки : "; cin >> newValue;  ThirdTask(L\_root, newValue);  break;  }  }  }  } |

Исходный код заголовочного файла NodeList.h

|  |
| --- |
| #pragma once  #include <iostream>  using namespace std;  struct Node  {  int value;  Node\* Next = nullptr;  };  void ListOutput(auto NodeList) {  if (NodeList->Next != nullptr) {  cout << NodeList->value << " -> ";  ListOutput(NodeList->Next);  }  else {  cout << NodeList->value;  }  }  void AddNewNode(auto& ListRoot, int newValue) {  if (ListRoot == nullptr) {  ListRoot = new Node{ newValue };  }  else {  Node\* current = ListRoot;  while (current->Next != nullptr) current = current->Next;  current->Next = new Node{ newValue };  }  }  bool FirstTask(auto& A, Node\* ListNode) {  while (ListNode != nullptr) {  Node\* curr = ListNode->Next;  while (curr != nullptr) {  if (ListNode->value == curr->value) {  return true;  }  curr = curr->Next;  }  ListNode = ListNode->Next;  }  return false;  }  void SecondTask(Node\*& NodeList) {  if (NodeList == nullptr) {  return;  }  Node\* maxNode = NodeList;  Node\* prevMaxNode = nullptr;  Node\* curr = NodeList->Next;  while (curr != nullptr) {  if (curr->value > maxNode->value) {  maxNode = curr;  prevMaxNode = NodeList;  }  else if (curr->value == maxNode->value) {  prevMaxNode = maxNode;  }  curr = curr->Next;  }  if (maxNode == NodeList) {  NodeList = NodeList->Next;  }  else if (prevMaxNode && prevMaxNode->Next != maxNode) {  prevMaxNode = prevMaxNode->Next;  }  if (prevMaxNode) {  prevMaxNode->Next = maxNode->Next;  }  else {  NodeList = maxNode->Next;  }  delete maxNode;  }  void ThirdTask(Node\*& L, int newValue) {  Node\* current = L;  int count = 1;  while (current != nullptr) {  if (count % 2 == 0) {  Node\* newNode = new Node;  newNode->value = newValue;  newNode->Next = current;  Node\* prev = L;  while (prev->Next != current) {  prev = prev->Next;  }  prev->Next = newNode;  }  current = current->Next;  count++;  }  } |