|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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-08-22 | Сенькевич Г.Д. |
| 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.

# **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.

Вариант №6. Условие задания:

|  |  |
| --- | --- |
| Упражнение 1 | Разработать функцию, которая вставляет перед последним узлом в списке два новых узла. |
| Упражнение 2 | Разработать функцию, которая удаляет из списка первое отрицательное значение, если оно присутствует в списке. |
| Упражнение 3 | Разработать функцию, которая находит в списке максимальное значение и перемещает его узел в конец списка. |

# **Решение**

Однонаправленный динамический список (также известный как односвязный список или просто связный список) является структурой данных, используемой в программировании для организации коллекции элементов, где каждый элемент содержит ссылку только на следующий элемент в списке, а последний элемент ссылается на NULL (или нулевой указатель), чтобы обозначить конец списка.

Однонаправленный динамический список состоит из узлов, где каждый узел содержит данные и ссылку (или указатель) на следующий узел. Одна из главных особенностей однонаправленного списка — это возможность динамического распределения памяти под новые элементы списка по мере необходимости, что делает его гибким и эффективным для работы с динамически изменяющимися данными.

Однонаправленные списки могут быть использованы для решения различных задач, таких как реализация стеков, очередей, поиска путей в графах и т. д. Однако, также есть и ограничения. Например, доступ к произвольному элементу в списке занимает O(n) времени, где n - количество элементов в списке, в отличие от массивов, где такой доступ может быть выполнен за O(1) времени. Однако, однонаправленные списки имеют преимущества, такие как более эффективное удаление и вставка элементов в середину списка, что делает их полезными в определенных сценариях.

Пример операций, которые можно выполнить со связным списком, включают:

* Вставка элемента в начало списка: создание нового узла, установка его ссылки на текущий первый элемент списка и обновление ссылки первого элемента на новый узел. Время выполнения: O(1).
* Вставка элемента в конец списка: создание нового узла, установка ссылки последнего элемента на новый узел. Время выполнения: O(n), где n - количество элементов в списке, так как требуется пройти весь список для нахождения последнего элемента.
* Удаление элемента из списка: поиск узла, содержащего данный элемент, обновление ссылок узлов перед и после найденного узла, освобождение памяти, занимаемой удаленным узлом. Время выполнения: O(n), где n - количество элементов в списке, так как может потребоваться пройти весь список для поиска удаляемого.

Ответ на вопрос №6 (Дайте определение структуре данных линейный список):

Линейный список — это структура данных, представляющая собой последовательность элементов, где каждый элемент содержит как сами данные, так и ссылку на следующий элемент списка. Элементы списка располагаются в определенном порядке, начиная с первого элемента и заканчивая последним элементом, который ссылается на нулевой элемент или на пустой список.

Линейный список обладает следующими свойствами:

* каждый элемент списка содержит данные и ссылку на следующий элемент (кроме последнего элемента, который ссылается на пустой список);
* элементы списка располагаются в определенном порядке, начиная с первого элемента и заканчивая последним элементом;
* размер списка может быть изменен путем добавления или удаления элементов.

Линейный список может быть однонаправленным, когда элементы могут быть просмотрены только в одном направлении, или двунаправленным, когда элементы могут быть просмотрены в обоих направлениях.

Линейный список является одним из наиболее распространенных типов структур данных и находит широкое применение в различных областях программирования, в том числе в алгоритмах обработки данных и управления памятью.

Для решения первого упражнения была написана функция insert\_two\_before\_last, выполняющая вставку двух новых узлов в список перед последним. Функция принимает на вход две целочисленные переменные – значения новых узлов и ничего не возвращает.

|  |
| --- |
| void list::insert\_two\_before\_last(double value1, double value2)  {  insert\_before(last, value1);  insert\_before(last, value2);  } |

Для решения второго упражнения была написана функция delete\_first\_negative, выполняющая удаление первого узла списка с отрицательным значением. Функция не принимает на вход аргументов и ничего не возвращает.

|  |
| --- |
| void list::delete\_first\_negative()  {  list\_node\* first\_negative = find\_first\_negative();  remove(first\_negative);  } |

Для решения третьего упражнения была написана функция move\_max\_to\_end, выполняющая поиск максимума в списке и перемещение его в конец. Функция не принимает на вход аргументов и ничего не возвращает.

|  |
| --- |
| void list::move\_max\_to\_end()  {  list\_node\* max = find\_max();  double max\_value = max->value;  remove(max);  insert\_after(last, max\_value);  } |

При запуске программы пользователь видит приглашение ввести элементы списка в консоль.

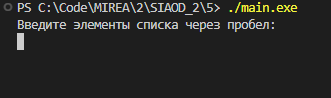


Рисунок 1. Интерфейс программы

# **Тестирование**

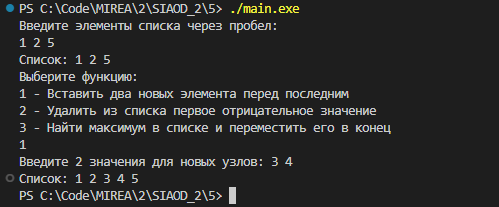
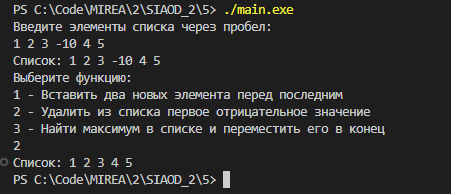
Протестируем выполнение программой первого упражнения. Для этого введём с клавиатуры значения 1, 2, 5, выберем в меню функцию 1 и введём значения новых узлов – 3 и 4; программа должна вывести в качестве результата список 1, 2, 3, 4, 5. На рисунке 2 видим, как программа вывела верный ответ.

Рисунок 2. Тестирование программы

Протестируем выполнение программой второго упражнения. Для этого введём с клавиатуры значения 1, 2, 3, -10, 4, 5 и выберем в меню функцию 2; программа должна вывести в качестве результата список 1, 2, 3, 4, 5. На рисунке 3 видим, как программа вывела верный ответ.

Рисунок 3. Тестирование программы

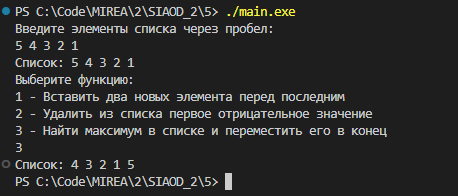
Протестируем выполнение программой третьего упражнения. Для этого введём с клавиатуры значения 5, 4, 3, 2, 1 и выберем в меню функцию 3; программа должна вывести в качестве результата список 4, 3, 2, 1, 5. На рисунке 4 видим, как программа вывела верный ответ.

Рисунок 4. Тестирование программы

Из результатов выполнения программы видно, что программа работает корректно, выполняя все поставленные задачи.

# **Вывод**

В результате выполнения работы я:

1. Научился реализовывать одноразрядный динамический список на языке программирования C++;
2. Освоил алгоритмы работы с однонаправленными динамическими списками.

# **Исходный код программы**

|  |
| --- |
| // ФАЙЛ main.cpp  #include <iostream>  #include "list.h"  int main()  {  list l;  std::cout << "Введите элементы списка через пробел:\n";  char pilot = std::cin.get();  double elem = 0;  while (pilot != '\n')  {  std::cin.putback(pilot);  std::cin >> elem;  l.append(elem);  pilot = std::cin.get();  }  while (true)  {  std::cout << "Список: ";  l.print();  int choice = 0;  std::cout << "Выберите функцию:\n"  << "1 - Вставить два новых элемента перед последним\n"  << "2 - Удалить из списка первое отрицательное значение\n"  << "3 - Найти максимум в списке и переместить его в конец\n"  << "4 - Выход\n";  std::cin >> choice;  if (choice == 1)  {  double value1 = 0, value2 = 0;  std::cout << "Введите 2 значения для новых узлов: ";  std::cin >> value1 >> value2;  l.insert\_two\_before\_last(value1, value2);  }  else if (choice == 2)  {  l.delete\_first\_negative();  }  else if (choice == 3)  {  l.move\_max\_to\_end();  }  else if (choice == 4)  {  break;  }  else  {  std::cout << "Такого варианта нет!\n";  }  }  return 0;  }  // ФАЙЛ list.h  #ifndef LIST\_H  #define LIST\_H  #include <iostream>  struct list\_node  {  double value;  list\_node\* next;  list\_node();  list\_node(double value);  };  struct list  {  list\_node\* first;  list\_node\* last;  int size;  list();  ~list();  void insert\_before(list\_node\* node, double value);  void insert\_after(list\_node\* node, double value);  void append(double value);  void remove(list\_node\* node);  void remove\_after(list\_node\* node);  void print();  list\_node\* find\_max();  list\_node\* find\_first\_negative();  void insert\_two\_before\_last(double value1, double value2);  void delete\_first\_negative();  void move\_max\_to\_end();  };  #endif  // ФАЙЛ list.cpp  #include "list.h"  list\_node::list\_node()  {  value = 0;  next = nullptr;  }  list\_node::list\_node(double value)  {  this->value = value;  next = nullptr;  }  list::list()  {  first = nullptr;  last = nullptr;  size = 0;  }  list::~list()  {  list\_node\* current = first;  while (current != nullptr)  {  list\_node\* next\_node = current->next;  delete current;  current = next\_node;  }  }  void list::insert\_before(list\_node\* node, double value)  {  if (node == nullptr)  {  insert\_after(last, value);  }  else if (node == first)  {  list\_node\* new\_node = new list\_node(value);  new\_node->next = first;  first = new\_node;  size++;  }  else  {  list\_node\* current = first;  while (current != nullptr)  {  if (current->next == node)  {  insert\_after(current, value);  break;  }  current = current->next;  }  }  }  void list::insert\_after(list\_node\* node, double value)  {  if (node == nullptr)  {  if (first != nullptr)  {  insert\_before(first, value);  }  else  {  first = new list\_node(value);  last = first;  size++;  }  }  else  {  list\_node\* new\_node = new list\_node(value);  new\_node->next = node->next;  node->next = new\_node;  size++;  if (last == node)  {  last = new\_node;  }  }  }  void list::append(double value)  {  insert\_after(last, value);  }  void list::remove(list\_node\* node)  {  if (node == first)  {  first = node->next;  delete node;  }  else  {  list\_node\* current = first;  while (current != nullptr)  {  if (current->next == node)  {  remove\_after(current);  break;  }  current = current->next;  }  }  }  void list::remove\_after(list\_node\* node)  {  if (node != nullptr && node->next != nullptr)  {  list\_node\* to\_delete = node->next;  node->next = to\_delete->next;  if (last == to\_delete)  {  last = node;  }  delete to\_delete;  }  }  void list::print()  {  list\_node\* current = first;  while (current != nullptr)  {  std::cout << current->value << ' ';  current = current->next;  }  std::cout << '\n';  }  list\_node\* list::find\_max()  {  list\_node\* max = nullptr;  list\_node\* current = first;  while (current != nullptr)  {  if (max == nullptr || current->value > max->value)  {  max = current;  }  current = current->next;  }  return max;  }  list\_node\* list::find\_first\_negative()  {  list\_node\* current = first;  while (current != nullptr)  {  if (current->value < 0)  {  return current;  }  current = current->next;  }  return nullptr;  }  void list::insert\_two\_before\_last(double value1, double value2)  {  insert\_before(last, value1);  insert\_before(last, value2);  }  void list::delete\_first\_negative()  {  list\_node\* first\_negative = find\_first\_negative();  remove(first\_negative);  }  void list::move\_max\_to\_end()  {  list\_node\* max = find\_max();  double max\_value = max->value;  remove(max);  insert\_after(last, max\_value);  } |