# Лабораторная работа №6

Выполнил: Лосев Данил

**Часть 1** (на основе лекции №6)

1. Реализовать коды функций с комментариями в удобном для Вас редакторе кода на языке программирования С++, используя структуру звена двунаправленного списка:

* void Postroenie (node \*\*nap, node \*\*ksp) - построение двунаправленного списка.
* void VyvodForward (node \*\*nsp, node \*\*ksp) – обход списка от его начала
* void VyvodForward (node \*\*nsp, node \*\*ksp) - обход списка от его конца

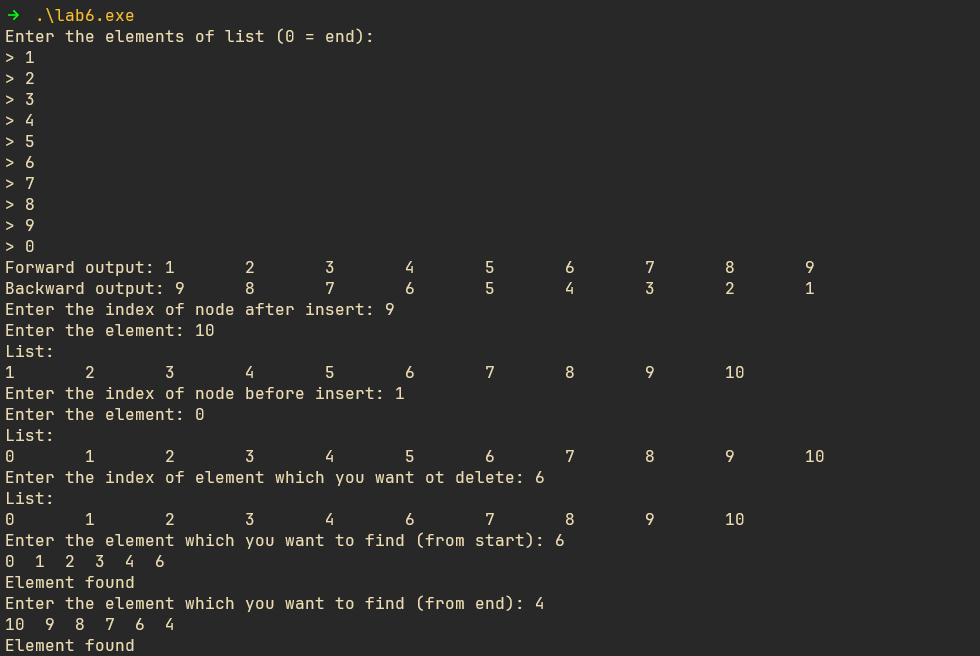
**Часть 2** (на основе лекции №7)

1. Реализовать коды функций с комментариями:

* Поиска звена по значению с начала списка
* Поиска звена с конца списка
* Функция вставки звена в список после звена, на которое указываетссылка
* Функция вставки звена в список перед звеном, на которое указываетссылка
* Удаление последнего звена из списка
* Удаление не последнего звена из списка

1. Все функции объединить в единый листинг с помощью меню.
2. Оформить отчет в электронном виде.

|  |
| --- |
| #include <cstddef>  #include <iostream>  // Структура, представляющая узел двусвязного списка  struct Node  {  int element; // Элемент, хранящийся в узле  Node \*next; // Указатель на следующий узел  Node \*previous; // Указатель на предыдущий узел  };  // Структура для представления самого двусвязного списка  struct List  {  Node \*start; // Указатель на начало списка  Node \*end; // Указатель на конец списка  };  // Прототипы функций для работы со списком  void buildingList(Node \*&startNode, Node \*&endNode); // Построение списка  void outputForward(Node \*startNode); // Вывод списка с начала до конца  void outputBackward(Node \*endNode); // Вывод списка с конца до начала  void insertAfter(Node \*&startNode, Node \*&endNode, int fIndex, int fElement); // Вставка узла после заданного индекса  void insertBefore(Node \*&startNode, Node \*&endNode, int fIndex, int fElement); // Вставка узла перед заданным индексом  void deleteNode(Node \*&startNode, Node \*&endNode, int fIndex); // Удаление узла по индексу  void deletingList(Node \*&startNode, Node \*&endNode); // Удаление всего списка  Node \*searchForward(Node \*startNode, int fElement); // Поиск элемента с начала списка  Node \*searchBackward(Node \*endNode, int fElement); // Поиск элемента с конца списка  // Главная функция программы  int main()  {  int index, element; // Переменные для хранения индекса и элемента  List myList; // Объявление списка  Node \*searchNode; // Указатель для хранения результата поиска  // Построение двусвязного списка  buildingList(myList.start, myList.end);  // Вывод списка по порядку  std::cout << "Forward output: ";  outputForward(myList.start);  // Вывод списка в обратном порядке  std::cout << "Backward output: ";  outputBackward(myList.end);  // Вставка узла после указанного индекса  std::cout << "Enter the index of node after insert: ";  std::cin >> index;  std::cout << "Enter the element: ";  std::cin >> element;  insertAfter(myList.start, myList.end, index, element);  std::cout << "List: \n";  outputForward(myList.start);  // Вставка узла перед указанным индексом  std::cout << "Enter the index of node before insert: ";  std::cin >> index;  std::cout << "Enter the element: ";  std::cin >> element;  insertBefore(myList.start, myList.end, index, element);  std::cout << "List: \n";  outputForward(myList.start);  // Удаление узла по индексу  std::cout << "Enter the index of element which you want ot delete: ";  std::cin >> index;  deleteNode(myList.start, myList.end, index);  std::cout << "List: \n";  outputForward(myList.start);  // Поиск элемента с начала списка  std::cout << "Enter the element which you want to find (from start): ";  std::cin >> element;  searchNode = searchForward(myList.start, element);  // Поиск элемента с конца списка  std::cout << "Enter the element which you want to find (from end): ";  std::cin >> element;  searchNode = searchBackward(myList.end, element);  // Удаление всего списка  deletingList(myList.start, myList.end);  return 0;  }  // Функция для построения списка  void buildingList(Node \*&startNode, Node \*&endNode)  {  int element; // Переменная для хранения вводимого элемента  Node \*curNode; // Текущий узел списка  startNode = new (Node); // Инициализация первого узла  curNode = startNode; // Присваивание текущему узлу начальный узел  (\*startNode).next = NULL; // Указатель на следующий узел у первого узла равен NULL  (\*startNode).previous = NULL; // Указатель на предыдущий узел у первого узла равен NULL  (\*startNode).element = 0; // Начальный элемент списка равен 0 (служебное значение)  std::cout << "Enter the elements of list (0 = end):" << '\n' << "> ";  std::cin >> element;  while (element != 0) // Пока введённый элемент не равен 0  {  (\*curNode).next = new (Node); // Создание нового узла и привязка его к текущему  (\*((\*curNode).next)).previous = curNode; // Присваивание предыдущего узла новому узлу  curNode = (\*curNode).next; // Переход на следующий узел  (\*curNode).next = NULL; // Указатель на следующий узел равен NULL  (\*curNode).element = element; // Присваивание элементу текущего узла введённое значение  endNode = curNode; // Обновление указателя на последний узел  std::cout << "> ";  std::cin >> element; // Ввод следующего элемента списка  }  }  // Функция для вывода списка с начала до конца  void outputForward(Node \*startNode)  {  Node \*curNode = (\*startNode).next; // Начинаю с первого узла после служебного  while (curNode != NULL) // Пока не достигнут конец списка  {  std::cout << (\*curNode).element << '\t'; // Вывод значения элемента узла  curNode = (\*curNode).next; // Переход к следующему узлу  }  std::cout << std::endl;  }  // Функция для вывода списка с конца к началу  void outputBackward(Node \*endNode)  {  Node \*curNode = endNode; // Начинаю с последнего узла списка  while ((\*curNode).previous != NULL) // Пока не достигнут первый узел  {  std::cout << (\*curNode).element << '\t'; // Вывод значения элемента узла  curNode = (\*curNode).previous; // Переход к предыдущему узлу  }  std::cout << std::endl;  }  // Функция для вставки узла после указанного индекса  void insertAfter(Node \*&startNode, Node \*&endNode, int fIndex, int fElement)  {  Node \*curNode = (\*startNode).next; // Начинаю с первого узла после служебного  Node \*newNode = new (Node); // Создаю новый узел для вставки  for (int i = 0; i < fIndex - 1; i++) // Перехожу на узел по указанному индексу  {  if ((\*curNode).next == NULL) // Если достигнут конец списка, прекращаю  {  break;  }  curNode = (\*curNode).next; // Переход на следующий узел  }  (\*newNode).element = fElement; // Присваиваю новому узлу значение элемента  if ((\*curNode).next != NULL) // Если узел не последний в списке  {  (\*newNode).next = (\*curNode).next; // Привязываю новый узел к следующему узлу  (\*newNode).previous = (\*(\*curNode).next).previous; // Присваиваю новый указатель на предыдущий узел  (\*(\*curNode).next).previous = newNode; // Обновляю указатель на новый узел у следующего узла  (\*curNode).next = newNode; // Привязываю новый узел к текущему узлу  }  else // Если узел последний  {  (\*newNode).next = NULL; // У нового узла нет следующего узла  (\*newNode).previous = curNode; // Устанавливаю предыдущий узел как текущий  endNode = newNode; // Обновляю указатель на последний узел  (\*curNode).next = newNode; // Привязываю новый узел к текущему  }  }  // Функция для вставки узла перед указанным индексом  void insertBefore(Node \*&startNode, Node \*&endNode, int fIndex, int fElement)  {  Node \*curNode = (\*startNode).next; // Начинаю с первого узла после служебного  Node \*newNode = new (Node); // Создаю новый узел для вставки  for (int i = 0; i < fIndex - 1; i++) // Перехожу на узел по указанному индексу  {  if ((\*curNode).next == NULL) // Если достигнут конец списка, прекращаю  {  break;  }  curNode = (\*curNode).next; // Переход на следующий узел  }  (\*newNode).element = fElement; // Присваиваю новому узлу значение элемента  (\*newNode).next = (\*(\*curNode).previous).next; // Привязываю новый узел к предыдущему  (\*newNode).previous = (\*curNode).previous; // Привязываю новый узел к следующему  (\*(\*curNode).previous).next = newNode; // Обновляю указатель у предыдущего узла  (\*curNode).previous = newNode; // Привязываю новый узел к текущему  }  // Функция для удаления узла по индексу  void deleteNode(Node \*&startNode, Node \*&endNode, int fIndex)  {  Node \*curNode = (\*startNode).next; // Начинаю с первого узла после служебного  for (int i = 0; i < fIndex - 1; i++) // Перехожу на узел по указанному индексу  {  if ((\*curNode).next == NULL) // Если достигнут конец списка, прекращаю  {  break;  }  curNode = (\*curNode).next; // Переход на следующий узел  }  if ((\*curNode).next != NULL) // Если узел не последний  {  (\*(\*curNode).next).previous = (\*curNode).previous; // Обновляю указатель предыдущего узла  (\*(\*curNode).previous).next = (\*curNode).next; // Обновляю указатель следующего узла  delete curNode; // Удаляю текущий узел  }  else // Если узел последний  {  (\*(\*curNode).previous).next = NULL; // У предыдущего узла больше нет следующего  endNode = (\*endNode).previous; // Обновляю указатель на последний узел  delete curNode; // Удаляю текущий узел  }  }  // Функция для поиска элемента с начала списка  Node \*searchForward(Node \*startNode, int fElement)  {  Node \*curNode = (\*startNode).next; // Начинаю с первого узла после служебного  Node \*foundNode = NULL; // Переменная для хранения результата поиска  while ((curNode != NULL) && (foundNode == NULL)) // Пока не найден элемент и не достигнут конец списка  {  std::cout << (\*curNode).element << " "; // Вывод значения текущего узла  if ((\*curNode).element == fElement) // Если элемент найден  {  std::cout << "\nElement found"; // Сообщение о нахождении элемента  foundNode = curNode; // Присваиваю найденный узел  }  else // Если элемент не найден  {  curNode = (\*curNode).next; // Переход к следующему узлу  }  }  if (foundNode == NULL) // Если элемент не найден  {  std::cout << "\nElement not found"; // Сообщение о ненахождении элемента  }  std::cout << std::endl;  return foundNode; // Возвращаю найденный узел или NULL  }  // Функция для поиска элемента с конца списка  Node \*searchBackward(Node \*endNode, int fElement)  {  Node \*curNode = endNode; // Начинаю с последнего узла списка  Node \*foundNode = NULL; // Переменная для хранения результата поиска  while (((\*curNode).previous != NULL) && (foundNode == NULL)) // Пока не найден элемент и не достигнут первый узел  {  std::cout << (\*curNode).element << " "; // Вывод значения текущего узла  if ((\*curNode).element == fElement) // Если элемент найден  {  std::cout << "\nElement found"; // Сообщение о нахождении элемента  foundNode = curNode; // Присваиваю найденный узел  }  else // Если элемент не найден  {  curNode = (\*curNode).previous; // Переход к предыдущему узлу  }  }  if (foundNode == NULL) // Если элемент не найден  {  std::cout << "\nElement not found"; // Сообщение о ненахождении элемента  }  std::cout << std::endl;  return foundNode; // Возвращаю найденный узел или NULL  }  // Функция для удаления всего списка  void deletingList(Node \*&startNode, Node \*&endNode)  {  Node \*curNode = startNode; // Начинаю с начального узла списка  Node \*nextNode = (\*curNode).next; // Переменная для хранения следующего узла  while ((\*curNode).next != NULL) // Пока не достигнут конец списка  {  delete curNode; // Удаляю текущий узел  curNode = nextNode; // Переход на следующий узел  nextNode = (\*curNode).next; // Обновляю указатель на следующий узел  }  delete curNode; // Удаляю последний узел  startNode = endNode = NULL; // Обнуляю указатели на начало и конец списка  } |



**Контрольные вопросы**

1. Что такое двусвязный список?

**Двусвязный список** — это динамическая структура данных, состоящая из узлов, где каждый узел хранит:

* **значение элемента**,
* **указатель на следующий узел**,
* **указатель на предыдущий узел**.

В двусвязном списке можно перемещаться как вперед, так и назад по элементам, благодаря двум указателям.

1. В чем отличие односвязных списков от двусвязных?

 В **односвязном списке** каждый узел хранит **только один указатель** — на **следующий узел**. Поэтому передвигаться можно только в одном направлении — вперед по списку.

 В **двусвязном списке** каждый узел хранит **два указателя** — один на **следующий узел** и другой на **предыдущий узел**, что позволяет передвигаться как вперед, так и назад.

1. Преимущества и недостатки двусвязного списка по сравнению с односвязным списком?

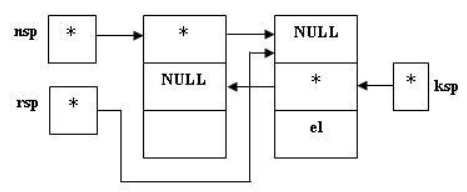
**Преимущества:**

* **Двусторонняя навигация**: можно легко двигаться по списку в обоих направлениях.
* **Упрощенное удаление узлов**: можно легко удалить узел, имея доступ к нему напрямую, не проходя весь список от начала.
* **Удобная вставка в середину**: легче вставлять новые элементы в середину, так как есть доступ к предыдущему и следующему элементам.

**Недостатки:**

* **Больший расход памяти**: каждый узел требует дополнительного указателя на предыдущий элемент, что увеличивает затраты памяти.
* **Сложность при изменении структуры**: нужно корректно обновлять оба указателя при вставке или удалении узлов.
* **Чуть более медленная работа**: из-за необходимости обработки двух указателей на каждом шаге операции могут быть немного медленнее, чем в односвязном списке.

1. Правильно ли настроили указатель ksp на конец списка ksp=rsp? нет



1. В каких направлениях можно обходить и осуществлять поиск звеньев в двусвязном списке?

В **двусвязном списке** можно обходить и осуществлять поиск в **двух направлениях**:

**Вперед** — начиная от **головы списка** (head) и перемещаясь по указателям на **следующий узел**. Это стандартное направление обхода, аналогичное односвязному списку.

**Назад** — начиная от **хвоста списка** (tail) и перемещаясь по указателям на **предыдущий узел**. Это возможно благодаря наличию обратных указателей, что отличает двусвязный список от односвязного.

1. Сколько  всего указателей используют при работе с двусвязным списком?

3

1. Сколько текущих указателей используют при работе с двусвязным списком?

1

1. Какие есть случаи вставки звена в двусвязный список? Отличие от односвязного списка?

### **Случаи вставки звена в двусвязный список:**

1. **Вставка в начало списка (перед первым узлом)**:
2. **Вставка в конец списка (после последнего узла)**:
3. **Вставка в середину списка (между двумя узлами)**:

### **Отличие от односвязного списка:**

* В **односвязном списке** при вставке необходимо обновить только **один указатель** — на следующий элемент (для предыдущего узла).
* В **двусвязном списке** при вставке нужно обновить **два указателя**:
  + У нового узла — указатели на предыдущий и следующий узлы.
  + У соседних узлов — указатели на новый узел.

Это делает процесс вставки в двусвязном списке несколько сложнее, но за счет этого обеспечивается более удобный доступ к элементам с обеих сторон списка.

1. Сколько дополнительных (текущих) указателей используют при вставке нового звена?

2

1. Сколько дополнительных (текущих) указателей используют при удалении звена?

1

|  |  |
| --- | --- |
| void buildingList(Node \*&startNode, Node \*&endNode)  {  int element; // Переменная для хранения вводимого элемента  Node \*curNode; // Текущий узел списка  startNode = new (Node); // Инициализация первого узла  curNode = startNode; // Присваивание текущему узлу начальный узел  (\*startNode).next = NULL; // Указатель на следующий узел у первого узла равен NULL  (\*startNode).previous = NULL; // Указатель на предыдущий узел у первого узла равен NULL  (\*startNode).element = 0; // Начальный элемент списка равен 0 (служебное значение)  std::cout << "Enter the elements of list (0 = end):" << '\n' << "> ";  std::cin >> element;  while (element != 0) // Пока введённый элемент не равен 0  {  (\*curNode).next = new (Node); // Создание нового узла и привязка его к текущему  (\*((\*curNode).next)).previous = curNode; // Присваивание предыдущего узла новому узлу  curNode = (\*curNode).next; // Переход на следующий узел  (\*curNode).next = NULL; // Указатель на следующий узел равен NULL  (\*curNode).element = element; // Присваивание элементу текущего узла введённое значение  endNode = curNode; // Обновление указателя на последний узел  std::cout << "> ";  std::cin >> element; // Ввод следующего элемента списка  }  } |  |
| void outputForward(Node \*startNode)  {  Node \*curNode = (\*startNode).next; // Начинаю с первого узла после служебного  while (curNode != NULL) // Пока не достигнут конец списка  {  std::cout << (\*curNode).element << '\t'; // Вывод значения элемента узла  curNode = (\*curNode).next; // Переход к следующему узлу  }  std::cout << std::endl;  } |  |
| void outputBackward(Node \*endNode)  {  Node \*curNode = endNode; // Начинаю с последнего узла списка  while ((\*curNode).previous != NULL) // Пока не достигнут первый узел  {  std::cout << (\*curNode).element << '\t'; // Вывод значения элемента узла  curNode = (\*curNode).previous; // Переход к предыдущему узлу  }  std::cout << std::endl;  } |  |
| void insertAfter(Node \*&startNode, Node \*&endNode, int fIndex, int fElement)  {  Node \*curNode = (\*startNode).next; // Начинаю с первого узла после служебного  Node \*newNode = new (Node); // Создаю новый узел для вставки  for (int i = 0; i < fIndex - 1; i++) // Перехожу на узел по указанному индексу  {  if ((\*curNode).next == NULL) // Если достигнут конец списка, прекращаю  {  break;  }  curNode = (\*curNode).next; // Переход на следующий узел  }  (\*newNode).element = fElement; // Присваиваю новому узлу значение элемента  if ((\*curNode).next != NULL) // Если узел не последний в списке  {  (\*newNode).next = (\*curNode).next; // Привязываю новый узел к следующему узлу  (\*newNode).previous = (\*(\*curNode).next).previous; // Присваиваю новый указатель на предыдущий узел  (\*(\*curNode).next).previous = newNode; // Обновляю указатель на новый узел у следующего узла  (\*curNode).next = newNode; // Привязываю новый узел к текущему узлу  }  else // Если узел последний  {  (\*newNode).next = NULL; // У нового узла нет следующего узла  (\*newNode).previous = curNode; // Устанавливаю предыдущий узел как текущий  endNode = newNode; // Обновляю указатель на последний узел  (\*curNode).next = newNode; // Привязываю новый узел к текущему  }  } |  |
| void insertBefore(Node \*&startNode, Node \*&endNode, int fIndex, int fElement)  {  Node \*curNode = (\*startNode).next; // Начинаю с первого узла после служебного  Node \*newNode = new (Node); // Создаю новый узел для вставки  for (int i = 0; i < fIndex - 1; i++) // Перехожу на узел по указанному индексу  {  if ((\*curNode).next == NULL) // Если достигнут конец списка, прекращаю  {  break;  }  curNode = (\*curNode).next; // Переход на следующий узел  }  (\*newNode).element = fElement; // Присваиваю новому узлу значение элемента  (\*newNode).next = (\*(\*curNode).previous).next; // Привязываю новый узел к предыдущему  (\*newNode).previous = (\*curNode).previous; // Привязываю новый узел к следующему  (\*(\*curNode).previous).next = newNode; // Обновляю указатель у предыдущего узла  (\*curNode).previous = newNode; // Привязываю новый узел к текущему  } |  |
| void deleteNode(Node \*&startNode, Node \*&endNode, int fIndex)  {  Node \*curNode = (\*startNode).next; // Начинаю с первого узла после служебного  for (int i = 0; i < fIndex - 1; i++) // Перехожу на узел по указанному индексу  {  if ((\*curNode).next == NULL) // Если достигнут конец списка, прекращаю  {  break;  }  curNode = (\*curNode).next; // Переход на следующий узел  }  if ((\*curNode).next != NULL) // Если узел не последний  {  (\*(\*curNode).next).previous = (\*curNode).previous; // Обновляю указатель предыдущего узла  (\*(\*curNode).previous).next = (\*curNode).next; // Обновляю указатель следующего узла  delete curNode; // Удаляю текущий узел  }  else // Если узел последний  {  (\*(\*curNode).previous).next = NULL; // У предыдущего узла больше нет следующего  endNode = (\*endNode).previous; // Обновляю указатель на последний узел  delete curNode; // Удаляю текущий узел  }  } |  |
| Node \*searchForward(Node \*startNode, int fElement)  {  Node \*curNode = (\*startNode).next; // Начинаю с первого узла после служебного  Node \*foundNode = NULL; // Переменная для хранения результата поиска  while ((curNode != NULL) && (foundNode == NULL)) // Пока не найден элемент и не достигнут конец списка  {  std::cout << (\*curNode).element << " "; // Вывод значения текущего узла  if ((\*curNode).element == fElement) // Если элемент найден  {  std::cout << "\nElement found"; // Сообщение о нахождении элемента  foundNode = curNode; // Присваиваю найденный узел  }  else // Если элемент не найден  {  curNode = (\*curNode).next; // Переход к следующему узлу  }  }  if (foundNode == NULL) // Если элемент не найден  {  std::cout << "\nElement not found"; // Сообщение о ненахождении элемента  }  std::cout << std::endl;  return foundNode; // Возвращаю найденный узел или NULL  } |  |
| Node \*searchBackward(Node \*endNode, int fElement)  {  Node \*curNode = endNode; // Начинаю с последнего узла списка  Node \*foundNode = NULL; // Переменная для хранения результата поиска  while (((\*curNode).previous != NULL) && (foundNode == NULL)) // Пока не найден элемент и не достигнут первый узел  {  std::cout << (\*curNode).element << " "; // Вывод значения текущего узла  if ((\*curNode).element == fElement) // Если элемент найден  {  std::cout << "\nElement found"; // Сообщение о нахождении элемента  foundNode = curNode; // Присваиваю найденный узел  }  else // Если элемент не найден  {  curNode = (\*curNode).previous; // Переход к предыдущему узлу  }  }  if (foundNode == NULL) // Если элемент не найден  {  std::cout << "\nElement not found"; // Сообщение о ненахождении элемента  }  std::cout << std::endl;  return foundNode; // Возвращаю найденный узел или NULL  } |  |
| void deletingList(Node \*&startNode, Node \*&endNode)  {  Node \*curNode = startNode; // Начинаю с начального узла списка  Node \*nextNode = (\*curNode).next; // Переменная для хранения следующего узла  while ((\*curNode).next != NULL) // Пока не достигнут конец списка  {  delete curNode; // Удаляю текущий узел  curNode = nextNode; // Переход на следующий узел  nextNode = (\*curNode).next; // Обновляю указатель на следующий узел  }  delete curNode; // Удаляю последний узел  startNode = endNode = NULL; // Обнуляю указатели на начало и конец списка  } |  |