**Белорусский государственный технологический университет**

**Факультет информационных технологий**

**Кафедра программной инженерии**

**Реферат “** **Реализация методов односвязного списка”**

По дисциплине «Основы алгоритмизации и программирования»

Выполнил:

Студент 1 курса 10 группы ПИ

Макаревич Кирилл Витальевич

Проверил: Белодед Николай Иванович

Оглавление

[**№1 Построение однонаправленного списка с заглавным звеном и его просмотр** 2](#_Toc196260893)

[Псевдокод: 3](#_Toc196260894)

[**№2 Включение звена с заданным элементом в однонаправленный список с заглавным звеном** 5](#_Toc196260895)

[Псевдокод: 8](#_Toc196260896)

[**№3 Удаление звена из однонаправленного списка с заглавным звеном.** 11](#_Toc196260897)

[Псевдокод: 14](#_Toc196260898)

# **№1 Построение однонаправленного списка с заглавным звеном и его просмотр**

|  |
| --- |
| #include <iostream>  using namespace std;  // определение класса Spisok для работы с однонаправленным списком  class Spisok {  private:  // приватная структура одного узла списка  struct node {  int elem; // элемент списка (целое число)  node\* sled; // указатель на следующий узел  } \*phead; // указатель на голову списка (заглавное звено)  public:  // конструктор класса Spisok  Spisok() {  phead = new node; // выделение памяти под голову  phead->sled = nullptr; // указатель на следующий элемент = null  }  // деструктор класса Spisok  ~Spisok() {  OCHISTKA(); // очистка всех элементов списка (кроме заглавного)  delete phead; // удаление головы  }  // метод для построения списка  void POSTROENIE();  // метод для вывода списка  void VYVOD();  // метод для очистки списка  void OCHISTKA();  };  // главная функция программы  int main() {  setlocale(LC\_ALL, "Russian");  Spisok A; // создание объекта A типа Spisok  A.POSTROENIE(); // вызов метода построения списка  A.VYVOD(); // вызов метода вывода списка  A.OCHISTKA(); // вызов метода очистки списка  cout << "\n"; // Переход на новую строку  return 0; // Завершение программы  }  // реализация метода POSTROENIE (создание списка)  void Spisok::POSTROENIE() {  node\* t = phead; // временный указатель, начинаем с головы  int el; // переменная для ввода значения  cout << "Вводите элементы списка (0 - конец): ";  cin >> el;  while (el != 0) { // цикл пока элемент не равен 0  t->sled = new node; // создание нового узла после текущего  t = t->sled; // переход к новому узлу  t->elem = el; // сохранение значения в новом узле  t->sled = nullptr; // следующий узел пока отсутствует  cin >> el; // ввод следующего значения  }  }  // реализация метода VYVOD (вывод списка на экран)  void Spisok::VYVOD() {  node\* t = phead->sled; // начинаем с первого реального элемента списка  cout << "Список: "; // вывод заголовка  while (t != nullptr) { // пока не дошли до конца списка  cout << t->elem << " "; // вывод текущего элемента  t = t->sled; // переход к следующему элементу  }  cout << endl;  }  // реализация метода OCHISTKA (удаление всех узлов списка)  void Spisok::OCHISTKA() {  node\* q = phead->sled; // указатель на первый реальный элемент  node\* q1; // вспомогательный указатель  while (q != nullptr) { // пока список не пуст  q1 = q->sled; // сохраняем указатель на следующий узел  delete q; // удаляем текущий узел  q = q1; // переходим к следующему узлу  }  phead->sled = nullptr; // указатель заглавного узла обнуляется  } |

## Псевдокод:

|  |
| --- |
| ОПРЕДЕЛИТЬ КЛАСС Spisok  ЗАКРЫТЫЙ РАЗДЕЛ:  ОПРЕДЕЛИТЬ СТРУКТУРУ node  ЦЕЛОЕ ЧИСЛО elem  УКАЗАТЕЛЬ НА node sled  КОНЕЦ СТРУКТУРЫ  ОБЪЯВИТЬ УКАЗАТЕЛЬ phead ТИПА node\*  ОТКРЫТЫЙ РАЗДЕЛ:  ОПРЕДЕЛИТЬ КОНСТРУКТОР Spisok()  ВЫДЕЛИТЬ ПАМЯТЬ ДЛЯ НОВОГО ЭЛЕМЕНТА node И ПРИСВОИТЬ phead  УСТАНОВИТЬ phead->sled В NULL  КОНЕЦ КОНСТРУКТОРА  ОПРЕДЕЛИТЬ ДЕСТРУКТОР ~Spisok()  ОСВОБОДИТЬ ПАМЯТЬ, УКАЗЫВАЕМУЮ phead  КОНЕЦ ДЕСТРУКТОРА  ОПРЕДЕЛИТЬ МЕТОД POSTROENIE()  ОБЪЯВИТЬ УКАЗАТЕЛЬ t ТИПА node\*  ОБЪЯВИТЬ ЦЕЛУЮ ПЕРЕМЕННУЮ el  ПРИСВОИТЬ t ← phead  ВЫВЕСТИ "Вводите элементы списка: "  СЧИТАТЬ el  ПОКА el НЕ РАВЕН 0 ВЫПОЛНЯТЬ:  ВЫДЕЛИТЬ НОВУЮ ПАМЯТЬ ДЛЯ ЭЛЕМЕНТА node И ПРИСВОИТЬ t->sled  ПЕРЕЙТИ К СЛЕДУЮЩЕМУ ЭЛЕМЕНТУ: t ← t->sled  ЗАДАТЬ t->elem ← el  ЗАДАТЬ t->sled ← NULL  СЧИТАТЬ el  КОНЕЦ ЦИКЛА  КОНЕЦ МЕТОДА  ОПРЕДЕЛИТЬ МЕТОД VYVOD()  ОБЪЯВИТЬ УКАЗАТЕЛЬ t ТИПА node\*  ПРИСВОИТЬ t ← phead->sled  ВЫВЕСТИ "Список: "  ПОКА t НЕ РАВЕН NULL ВЫПОЛНЯТЬ:  ВЫВЕСТИ t->elem С ПРОБЕЛОМ  ПЕРЕЙТИ К СЛЕДУЮЩЕМУ ЭЛЕМЕНТУ: t ← t->sled  КОНЕЦ ЦИКЛА  ПЕРЕЙТИ НА СЛЕДУЮЩУЮ СТРОКУ  КОНЕЦ МЕТОДА  ОПРЕДЕЛИТЬ МЕТОД OCHISTKA()  ОБЪЯВИТЬ УКАЗАТЕЛИ q И q1 ТИПА node\*  ПРИСВОИТЬ q ← phead  ПРИСВОИТЬ q1 ← q->sled  ПОКА q1 НЕ РАВЕН NULL ВЫПОЛНЯТЬ:  ПРИСВОИТЬ q ← q1  ПРИСВОИТЬ q1 ← q1->sled  УДАЛИТЬ ЭЛЕМЕНТ q ИЗ ПАМЯТИ  КОНЕЦ ЦИКЛА  КОНЕЦ МЕТОДА  КОНЕЦ КЛАССА  НАЧАЛО ПРОГРАММЫ  УСТАНОВИТЬ ЯЗЫКОВУЮ ЛОКАЛЬ ПРИ ЗАПУСКЕ НА "Русский"  СОЗДАТЬ ОБЪЕКТ A ТИПА Spisok  ВЫЗВАТЬ A.POSTROENIE() // построить список  ВЫЗВАТЬ A.VYVOD() // вывести элементы списка  ВЫЗВАТЬ A.OCHISTKA() // очистить память от списка  ВЫВЕСТИ ПУСТУЮ СТРОКУ  ВЫПОЛНИТЬ ПАУЗУ ДО НАЖАТИЯ КЛАВИШИ ПОЛЬЗОВАТЕЛЕМ  КОНЕЦ ПРОГРАММЫ |

# **№2 Включение звена с заданным элементом в однонаправленный список с заглавным звеном**

|  |
| --- |
| #include<iostream>  using namespace std;  struct node // определение структуры для элемента списка  {  int elem; // переменная для хранения значения элемента  node\* sled; // указатель на следующий элемент в списке  };  class Spisok // определение класса для работы с однонаправленным списком  {  private:  node\* phead; // указатель на начало списка (заглавное звено)  node\* Res; // указатель на найденное звено списка  public:  Spisok() { phead = new(node); (\*phead).sled = NULL; Res = NULL; } // конструктор, выделяющий память для головы (заглавного звена)  ~Spisok() { delete phead; } // деструктор, удаляющий память, выделенную для головы  void POSTROENIE(); // метод для построения списка  void VYVOD(); // метод для вывода списка  node\* POISK(int); // метод для поиска звена по значению  void VSTAV(int); // метод для вставки звена после найденного  void VSTAV1(int); // метод для вставки звена перед найденным  void OCHISTKA(); // метод для очистки списка из памяти  };  void main() // главная функция программы  {  setlocale(LC\_ALL, "Rus");  Spisok A; // создаем объект класса Spisok  int el, el1; // переменные для хранения значений элементов  A.POSTROENIE(); // строим список  A.VYVOD(); // выводим список  cout << "\nВведите элемент звена, после которого ";  cout << "осуществляется вставка:\n"; // печатаем для пользователя  cin >> el; // вводим элемент для поиска в списке  cout << "Введите элемент вставляемого звена:\n";  cin >> el1; // вводим элемент для вставки в список  if (A.POISK(el) != NULL) // если элемент найден в списке  {  A.VSTAV(el1); A.VYVOD(); // вставляем элемент после найденного и выводим список  }  else cout << "Звена с заданным элементом в списке нет!"; // если элемент не найден  cout << "\nВведите элемент звена, перед которым ";  cout << "осуществляется вставка:\n"; // печатаем для пользователя  cin >> el; // вводим элемент для поиска в списке  cout << " Введите элемент вставляемого звена:\n";  cin >> el1; // вводим элемент для вставки в список  if (A.POISK(el) != NULL) // если элемент найден в списке  {  A.VSTAV1(el1); A.VYVOD(); // вставляем элемент перед найденным и выводим список  }  else cout << " Звена с заданным элементом в списке нет!"; // если элемент не найден  A.OCHISTKA(); // очищаем список из памяти  cout << "\n"; // печатаем пустую строку  system("PAUSE"); // ожидаем нажатия клавиши для завершения  }  void Spisok::POSTROENIE() // метод для построения однонаправленного списка  {  node\* t; // указатель на текущий элемент списка  int el; // переменная для хранения ввода элемента  t = phead; // инициализация указателя на заглавное звено  cout << "Вводите элементы звеньев списка: ";  cin >> el; // вводим первый элемент  while (el != 0) // пока введенный элемент не равен 0  {  (\*t).sled = new (node); // выделяем память для нового элемента  t = (\*t).sled; (\*t).elem = el; (\*t).sled = NULL; // настраиваем указатели и сохраняем элемент  cin >> el; // вводим следующий элемент  }  }  void Spisok::VYVOD() // метод для вывода содержимого списка  {  node\* t; // указатель на текущий элемент списка  t = (\*phead).sled; // инициализация указателя на первый элемент  cout << "Список: "; // печатаем голову  while (t != NULL) // пока не достигнут конец списка  {  cout << (\*t).elem << " "; // печатаем текущий элемент  t = (\*t).sled; // переходим к следующему элементу  }  cout << endl; // печатаем новую строку  }  node\* Spisok::POISK(int el) // метод для поиска элемента в списке  {  node\* t; // указатель на текущий элемент списка  Res = NULL; // инициализация указателя на найденный элемент  t = (\*phead).sled; // инициализация указателя на первый элемент  while (t != NULL && Res == NULL) // пока не достигнут конец списка и элемент не найден  if ((\*t).elem == el) // если найден элемент  Res = t; // сохраняем указатель на найденный элемент  else t = (\*t).sled; // иначе переходим к следующему элементу  return Res; // возвращаем указатель на найденный элемент или NULL  }  void Spisok::VSTAV(int el) // метод для вставки элемента после найденного  {  node\* q; // указатель на новый элемент  q = new(node); // выделяем память для нового элемента  (\*q).elem = el; // сохраняем значение в новый элемент  (\*q).sled = (\*Res).sled; // новый элемент будет указывать на тот же элемент, на который указывает Res  (\*Res).sled = q; // устанавливаем указатель в найденном элементе на новый элемент  }  void Spisok::VSTAV1(int el) // метод для вставки элемента перед найденным  {  node\* q; // указатель на новый элемент  q = new (node); // выделяем память для нового элемента  (\*q).elem = (\*Res).elem; // сохраняем значение текущего элемента в новый элемент  (\*q).sled = (\*Res).sled; // новый элемент будет указывать на следующий элемент  (\*Res).elem = el; // сохраняем значение для текущего элемента  (\*Res).sled = q; // устанавливаем указатель в найденном элементе на новый элемент  }  void Spisok::OCHISTKA() // метод для удаления списка из памяти  {  node\* q, \* q1; // указатели для работы с элементами  q = phead; // инициализация указателя на заглавное звено  q1 = (\*q).sled; // указатель "опережает" указатель q  while (q1 != NULL) // пока не достигнут конец списка  {  q = q1; // переходим к следующему элементу  q1 = (\*q1).sled; // обновляем указатель на следующий элемент  delete q; // удаляем текущий элемент из памяти  }  } |

## Псевдокод:

|  |
| --- |
| ОПРЕДЕЛИТЬ СТРУКТУРУ node  ЦЕЛОЕ ЧИСЛО elem // переменная для хранения значения элемента  УКАЗАТЕЛЬ НА node sled // указатель на следующий элемент в списке  КОНЕЦ СТРУКТУРЫ  ОПРЕДЕЛИТЬ КЛАСС Spisok  ЗАКРЫТЫЙ РАЗДЕЛ:  ОБЪЯВИТЬ УКАЗАТЕЛЬ phead ТИПА node\* // указатель на начало списка  ОБЪЯВИТЬ УКАЗАТЕЛЬ Res ТИПА node\* // указатель на найденное звено в списке  ОТКРЫТЫЙ РАЗДЕЛ:  ОПРЕДЕЛИТЬ КОНСТРУКТОР Spisok()  ВЫДЕЛИТЬ ПАМЯТЬ ДЛЯ НОВОГО ЭЛЕМЕНТА node И ПРИСВОИТЬ phead  УСТАНОВИТЬ phead->sled В NULL  УСТАНОВИТЬ Res В NULL  КОНЕЦ КОНСТРУКТОРА  ОПРЕДЕЛИТЬ ДЕСТРУКТОР ~Spisok()  ОСВОБОДИТЬ ПАМЯТЬ, УКАЗЫВАЕМУЮ phead  КОНЕЦ ДЕСТРУКТОРА  ОПРЕДЕЛИТЬ МЕТОД POSTROENIE()  ОБЪЯВИТЬ УКАЗАТЕЛЬ t ТИПА node\*  ОБЪЯВИТЬ ЦЕЛУЮ ПЕРЕМЕННУЮ el  ПРИСВОИТЬ t ← phead  ВЫВЕСТИ "Вводите элементы списка: "  СЧИТАТЬ el  ПОКА el НЕ РАВЕН 0 ВЫПОЛНЯТЬ:  ВЫДЕЛИТЬ НОВУЮ ПАМЯТЬ ДЛЯ ЭЛЕМЕНТА node И ПРИСВОИТЬ t->sled  ПЕРЕЙТИ К СЛЕДУЮЩЕМУ ЭЛЕМЕНТУ: t ← t->sled  ЗАДАТЬ t->elem ← el  ЗАДАТЬ t->sled ← NULL  СЧИТАТЬ el  КОНЕЦ ЦИКЛА  КОНЕЦ МЕТОДА  ОПРЕДЕЛИТЬ МЕТОД VYVOD()  ОБЪЯВИТЬ УКАЗАТЕЛЬ t ТИПА node\*  ПРИСВОИТЬ t ← phead->sled  ВЫВЕСТИ "Список: "  ПОКА t НЕ РАВЕН NULL ВЫПОЛНЯТЬ:  ВЫВЕСТИ t->elem С ПРОБЕЛОМ  ПЕРЕЙТИ К СЛЕДУЮЩЕМУ ЭЛЕМЕНТУ: t ← t->sled  КОНЕЦ ЦИКЛА  ПЕРЕЙТИ НА СЛЕДУЮЩУЮ СТРОКУ  КОНЕЦ МЕТОДА  ОПРЕДЕЛИТЬ МЕТОД POISK(int el)  ОБЪЯВИТЬ УКАЗАТЕЛЬ t ТИПА node\*  Res ← NULL  ПРИСВОИТЬ t ← phead->sled  ПОКА t НЕ РАВЕН NULL И Res РАВЕН NULL ВЫПОЛНЯТЬ:  ЕСЛИ t->elem РАВЕН el ТО  Res ← t  ИНАЧЕ  t ← t->sled  КОНЕЦ ПОКА  ВЕРНУТЬ Res  КОНЕЦ МЕТОДА  ОПРЕДЕЛИТЬ МЕТОД VSTAV(int el)  ОБЪЯВИТЬ УКАЗАТЕЛЬ q ТИПА node\*  ВЫДЕЛИТЬ ПАМЯТЬ ДЛЯ ЭЛЕМЕНТА node И ПРИСВОИТЬ q  q->elem ← el  q->sled ← Res->sled  Res->sled ← q  КОНЕЦ МЕТОДА  ОПРЕДЕЛИТЬ МЕТОД VSTAV1(int el)  ОБЪЯВИТЬ УКАЗАТЕЛЬ q ТИПА node\*  ВЫДЕЛИТЬ ПАМЯТЬ ДЛЯ ЭЛЕМЕНТА node И ПРИСВОИТЬ q  q->elem ← Res->elem  q->sled ← Res->sled  Res->elem ← el  Res->sled ← q  КОНЕЦ МЕТОДА  ОПРЕДЕЛИТЬ МЕТОД OCHISTKA()  ОБЪЯВИТЬ УКАЗАТЕЛИ q И q1 ТИПА node\*  ПРИСВОИТЬ q ← phead  ПРИСВОИТЬ q1 ← q->sled  ПОКА q1 НЕ РАВЕН NULL ВЫПОЛНЯТЬ:  ПРИСВОИТЬ q ← q1  ПРИСВОИТЬ q1 ← q1->sled  УДАЛИТЬ q  КОНЕЦ ЦИКЛА  КОНЕЦ МЕТОДА  КОНЕЦ КЛАССА  НАЧАЛО ПРОГРАММЫ  УСТАНОВИТЬ ЯЗЫКОВУЮ ЛОКАЛЬ ПРИ ЗАПУСКЕ НА "Русский"  СОЗДАТЬ ОБЪЕКТ A ТИПА Spisok  A.POSTROENIE() // строим список  A.VYVOD() // выводим список  ВЫВЕСТИ "Введите элемент звена, после которого осуществляется вставка:"  СЧИТАТЬ el  ВЫВЕСТИ "Введите элемент вставляемого звена:"  СЧИТАТЬ el1  ЕСЛИ A.POISK(el) НЕ РАВЕН NULL ТО  A.VSTAV(el1)  A.VYVOD()  ИНАЧЕ  ВЫВЕСТИ "Звена с заданным элементом в списке нет!"  КОНЕЦ ЕСЛИ  ВЫВЕСТИ "Введите элемент звена, перед которым осуществляется вставка:"  СЧИТАТЬ el  ВЫВЕСТИ "Введите элемент вставляемого звена:"  СЧИТАТЬ el1  ЕСЛИ A.POISK(el) НЕ РАВЕН NULL ТО  A.VSTAV1(el1)  A.VYVOD()  ИНАЧЕ  ВЫВЕСТИ "Звена с заданным элементом в списке нет!"  КОНЕЦ ЕСЛИ  A.OCHISTKA() // очищаем список из памяти  ВЫВЕСТИ ПУСТУЮ СТРОКУ  ВЫПОЛНИТЬ ПАУЗУ ДО НАЖАТИЯ КЛАВИШИ ПОЛЬЗОВАТЕЛЕМ  КОНЕЦ ПРОГРАММЫ |

# **№3 Удаление звена из однонаправленного списка с заглавным звеном.**

|  |
| --- |
| #include<iostream>  using namespace std;  struct node  {  int elem; // хранит значение элемента  node\* sled; // указатель на следующее звено в списке  };  class Spisok {  private:  node\* phead, \* Res; // phead - указатель на начало списка (заглавное звено), Res - указатель на найденное звено  public:  Spisok() { phead = new(node); Res = NULL; } // конструктор: выделение памяти для заглавного звена  ~Spisok() { delete phead; } // деструктор: освобождение памяти для заглавного звена  void POSTROENIE(); // метод для построения списка  void VYVOD(); // метод для вывода списка  node\* POISK(int); // метод для поиска элемента в списке  void YDALE(); // метод для удаления элемента после найденного  void YDALE1(); // метод для удаления найденного элемента  void OCHISTKA(); // метод для очистки списка из памяти  };  void main()  {  setlocale(LC\_ALL, "Rus"); // установка русской локализации  Spisok A; // создание объекта A типа Spisok  int el; // переменная для хранения вводимого элемента  node\* Res\_Zn; // переменная для хранения результата поиска  A.POSTROENIE(); // построение списка  A.VYVOD(); // вывод списка  cout << "\nВведите элемент звена, после которого ";  cout << "осуществляется удаление:\n";  cin >> el; // ввод элемента для удаления  Res\_Zn = A.POISK(el); // поиск звена с данным элементом  if (Res\_Zn != NULL && (\*Res\_Zn).sled != NULL) // если звено найдено и не последнее  {  A.YDALE(); // удаление звена после найденного  A.VYVOD(); // вывод списка после удаления  }  else cout << "Звена с заданным элементом в списке нет!"; // если звено не найдено  cout << "\nВведите удаляемый элемент звена:\n";  cin >> el; // ввод элемента для удаления  Res\_Zn = A.POISK(el); // поиск элемента  if (A.POISK(el) != NULL) // если элемент найден  {  A.YDALE1(); // удаление найденного элемента  A.VYVOD(); // вывод списка после удаления  cout << endl;  }  else cout << "Звена с заданным элементом в списке нет!"; // если элемент не найден  A.OCHISTKA(); // очистка списка из памяти  cout << "\n";  system("PAUSE"); // пауза до нажатия клавиши  }  void Spisok::POSTROENIE()  // построение однонаправленного списка с заглавным звеном.  // phead - указатель на заглавное звено  {  node\* t; // указатель для обхода списка  int el; // вводимый элемент  t = phead; (\*t).sled = NULL; // инициализация заглавного звена  cout << "Вводите элементы звеньев списка: ";  cin >> el; // ввод первого элемента  while (el != 0) // пока не введено 0  {  (\*t).sled = new (node); // выделение памяти для нового элемента  t = (\*t).sled; // переход к новому элементу  (\*t).elem = el; // присваиваем значение элементу  (\*t).sled = NULL; // устанавливаем указатель на NULL  cin >> el; // ввод следующего элемента  }  }  void Spisok::VYVOD()  // вывод содержимого однонаправленного списка.  // phead - указатель на заглавное звено.  {  node\* t; // указатель для обхода списка  t = phead; t = (\*t).sled; // начинаем с первого элемента  cout << "Список: ";  while (t != NULL) // пока не дошли до конца списка  {  cout << (\*t).elem << " "; // выводим значение элемента  t = (\*t).sled; // переходим к следующему элементу  }  }  node\* Spisok::POISK(int el)  // поиск звена с элементом el в списке, заданном указателем phead.  // в случае успешного поиска в Res находится адрес искомого звена  // списка. В противном случае Res содержит NULL.  {  node\* t; // указатель для обхода списка  Res = NULL; // инициализация результата поиска  t = phead; // начинаем с заглавного звена  t = (\*t).sled; // переходим к первому элементу  while (t != NULL && Res == NULL) // пока не дошли до конца и не нашли элемент  if ((\*t).elem == el) // если элемент найден  Res = t; // присваиваем результат поиска  else  t = (\*t).sled; // переходим к следующему элементу  return Res; // возвращаем результат  }  void Spisok::YDALE()  // удаление звена, расположенного после звена,  // на которое указывает ссылка Res.  {  node\* q; // временный указатель для удаления  q = (\*Res).sled; // получаем указатель на следующий элемент  if (q != NULL) // если следующий элемент существует  {  (\*Res).sled = (\*(\*Res).sled).sled; // перенаправляем указатель на следующий элемент  delete q; // освобождаем память для удаляемого элемента  }  else  cout << "Звено с заданным элементом - последнее!\n"; // если элемент последний  }  void Spisok::YDALE1()  // удаление звена, на которое указывает ссылка Res.  {  node\* q, \* q1, \* q2; // временные указатели  q = (\*Res).sled; // получаем указатель на следующий элемент  if (q != NULL) // если следующий элемент существует  {  (\*Res).elem = (\*q).elem; // копируем значение из следующего элемента  (\*Res).sled = (\*q).sled; // перенаправляем указатель на следующий элемент  delete q; // освобождаем память для удаляемого элемента  }  else // если элемент последний  {  q1 = phead; // начинаем с заглавного звена  q2 = (\*q1).sled; // переходим к первому элементу  while (q2 != Res) // ищем элемент, перед которым нужно удалить  {  q1 = q2; // переходим к следующему элементу  q2 = (\*q2).sled;  }  (\*q1).sled = NULL; // перенаправляем указатель на NULL  q2 = NULL; // освобождаем указатель  delete Res; // удаляем текущий элемент  }  }  void Spisok::OCHISTKA()  // удаление однонаправленного списка из памяти.  // phead - указатель на заглавное звено списка.  {  node\* q, \* q1; // рабочие указатели  q = phead; // начинаем с заглавного звена  q1 = (\*q).sled; // указатель q1 "опережает" указатель q  while (q1 != NULL) // пока не дошли до конца списка  {  q = q1; // переходим к следующему элементу  q1 = (\*q1).sled; // переходим к следующему элементу  delete q; // освобождаем память для текущего элемента  }  } |

## Псевдокод:

|  |
| --- |
| ОПРЕДЕЛИТЬ СТРУКТУРУ node  ЦЕЛОЕ ЧИСЛО elem // переменная для хранения значения элемента  УКАЗАТЕЛЬ НА node sled // указатель на следующий элемент в списке  КОНЕЦ СТРУКТУРЫ  ОПРЕДЕЛИТЬ КЛАСС Spisok  ЗАКРЫТЫЙ РАЗДЕЛ:  ОБЪЯВИТЬ УКАЗАТЕЛЬ phead ТИПА node\* // указатель на начало списка  ОБЪЯВИТЬ УКАЗАТЕЛЬ Res ТИПА node\* // указатель на найденное звено в списке  ОТКРЫТЫЙ РАЗДЕЛ:  ОПРЕДЕЛИТЬ КОНСТРУКТОР Spisok()  ВЫДЕЛИТЬ ПАМЯТЬ ДЛЯ НОВОГО ЭЛЕМЕНТА node И ПРИСВОИТЬ phead  УСТАНОВИТЬ phead->sled В NULL  УСТАНОВИТЬ Res В NULL  КОНЕЦ КОНСТРУКТОРА  ОПРЕДЕЛИТЬ ДЕСТРУКТОР ~Spisok()  ОСВОБОДИТЬ ПАМЯТЬ, УКАЗЫВАЕМУЮ phead  КОНЕЦ ДЕСТРУКТОРА  ОПРЕДЕЛИТЬ МЕТОД POSTROENIE()  ОБЪЯВИТЬ УКАЗАТЕЛЬ t ТИПА node\*  ОБЪЯВИТЬ ЦЕЛУЮ ПЕРЕМЕННУЮ el  ПРИСВОИТЬ t ← phead  ВЫВЕСТИ "Вводите элементы списка: "  СЧИТАТЬ el  ПОКА el НЕ РАВЕН 0 ВЫПОЛНЯТЬ:  ВЫДЕЛИТЬ НОВУЮ ПАМЯТЬ ДЛЯ ЭЛЕМЕНТА node И ПРИСВОИТЬ t->sled  ПЕРЕЙТИ К СЛЕДУЮЩЕМУ ЭЛЕМЕНТУ: t ← t->sled  ЗАДАТЬ t->elem ← el  ЗАДАТЬ t->sled ← NULL  СЧИТАТЬ el  КОНЕЦ ЦИКЛА  КОНЕЦ МЕТОДА  ОПРЕДЕЛИТЬ МЕТОД VYVOD()  ОБЪЯВИТЬ УКАЗАТЕЛЬ t ТИПА node\*  ПРИСВОИТЬ t ← phead->sled  ВЫВЕСТИ "Список: "  ПОКА t НЕ РАВЕН NULL ВЫПОЛНЯТЬ:  ВЫВЕСТИ t->elem С ПРОБЕЛОМ  ПЕРЕЙТИ К СЛЕДУЮЩЕМУ ЭЛЕМЕНТУ: t ← t->sled  КОНЕЦ ЦИКЛА  ПЕРЕЙТИ НА СЛЕДУЮЩУЮ СТРОКУ  КОНЕЦ МЕТОДА  ОПРЕДЕЛИТЬ МЕТОД POISK(int el)  ОБЪЯВИТЬ УКАЗАТЕЛЬ t ТИПА node\*  Res ← NULL  ПРИСВОИТЬ t ← phead->sled  ПОКА t НЕ РАВЕН NULL И Res РАВЕН NULL ВЫПОЛНЯТЬ:  ЕСЛИ t->elem РАВЕН el ТО  Res ← t  ИНАЧЕ  t ← t->sled  КОНЕЦ ПОКА  ВЕРНУТЬ Res  КОНЕЦ МЕТОДА  ОПРЕДЕЛИТЬ МЕТОД VSTAV(int el)  ОБЪЯВИТЬ УКАЗАТЕЛЬ q ТИПА node\*  ВЫДЕЛИТЬ ПАМЯТЬ ДЛЯ ЭЛЕМЕНТА node И ПРИСВОИТЬ q  q->elem ← el  q->sled ← Res->sled  Res->sled ← q  КОНЕЦ МЕТОДА  ОПРЕДЕЛИТЬ МЕТОД VSTAV1(int el)  ОБЪЯВИТЬ УКАЗАТЕЛЬ q ТИПА node\*  ВЫДЕЛИТЬ ПАМЯТЬ ДЛЯ ЭЛЕМЕНТА node И ПРИСВОИТЬ q  q->elem ← Res->elem  q->sled ← Res->sled  Res->elem ← el  Res->sled ← q  КОНЕЦ МЕТОДА  ОПРЕДЕЛИТЬ МЕТОД OCHISTKA()  ОБЪЯВИТЬ УКАЗАТЕЛИ q И q1 ТИПА node\*  ПРИСВОИТЬ q ← phead  ПРИСВОИТЬ q1 ← q->sled  ПОКА q1 НЕ РАВЕН NULL ВЫПОЛНЯТЬ:  ПРИСВОИТЬ q ← q1  ПРИСВОИТЬ q1 ← q1->sled  УДАЛИТЬ q  КОНЕЦ ЦИКЛА  КОНЕЦ МЕТОДА  КОНЕЦ КЛАССА  НАЧАЛО ПРОГРАММЫ  УСТАНОВИТЬ ЯЗЫКОВУЮ ЛОКАЛЬ ПРИ ЗАПУСКЕ НА "Русский"  СОЗДАТЬ ОБЪЕКТ A ТИПА Spisok  A.POSTROENIE() // строим список  A.VYVOD() // выводим список  ВЫВЕСТИ "Введите элемент звена, после которого осуществляется вставка:"  СЧИТАТЬ el  ВЫВЕСТИ "Введите элемент вставляемого звена:"  СЧИТАТЬ el1  ЕСЛИ A.POISK(el) НЕ РАВЕН NULL ТО  A.VSTAV(el1)  A.VYVOD()  ИНАЧЕ  ВЫВЕСТИ "Звена с заданным элементом в списке нет!"  КОНЕЦ ЕСЛИ  ВЫВЕСТИ "Введите элемент звена, перед которым осуществляется вставка:"  СЧИТАТЬ el  ВЫВЕСТИ "Введите элемент вставляемого звена:"  СЧИТАТЬ el1  ЕСЛИ A.POISK(el) НЕ РАВЕН NULL ТО  A.VSTAV1(el1)  A.VYVOD()  ИНАЧЕ  ВЫВЕСТИ "Звена с заданным элементом в списке нет!"  КОНЕЦ ЕСЛИ  A.OCHISTKA() // очищаем список из памяти  ВЫВЕСТИ ПУСТУЮ СТРОКУ  ВЫПОЛНИТЬ ПАУЗУ ДО НАЖАТИЯ КЛАВИШИ ПОЛЬЗОВАТЕЛЕМ  КОНЕЦ ПРОГРАММЫ |