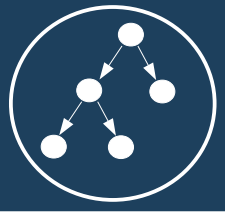


Data Structures and Algorithms

Структуры данных.
Двусвязный список

Двусвязный список

Двусвязный список — разновидность связанного списка. Узел содержит данные и две ссылки (указатели) на предыдущий и следующий элемент списка.

Преимущества и недостатки двусвязных списков

Преимущества:

- Простота вставки и удаления элемента в начале и конце списка

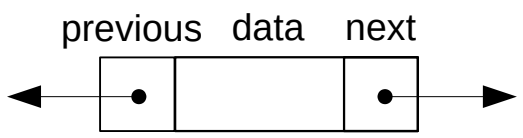
Недостатки:

- Сложность получения элемента по индексу
- Удвоенный (по сравнению с односвязным списком) расход памяти на хранение указателей

Узел списка

Узел списка представляет собой составную структуру. Обычно реализуется с помощью класса или структуры (в процедурных языках). Содержит значения двух типов. Одно для хранения данных (числа, строки и т. д.), и две ссылки на следующий узел или предыдущий узел(реализуется как указать или ссылка тип которых совпадает с типом узел).

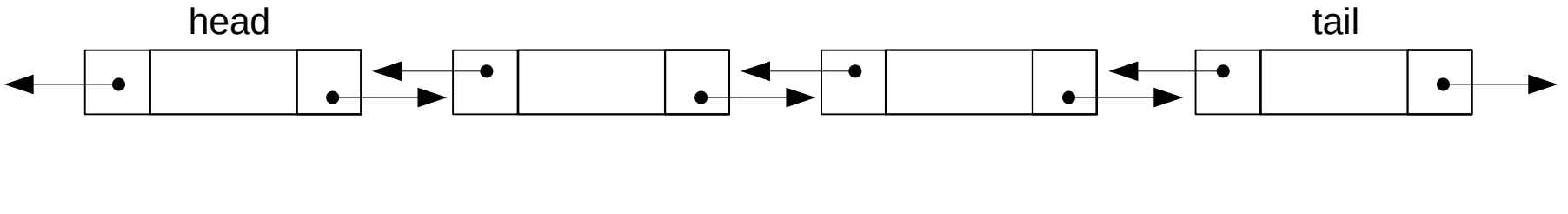
Схематическое
изображение узла списка



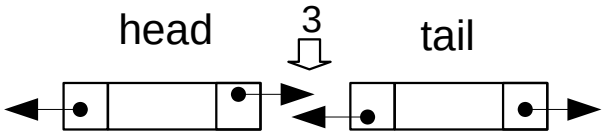
Двусвязный список

В одной из самых распространенных реализаций двусвязного используется два фиктивных элемента head и tail (начало и конец списка соответственно). Остальные узлы вставляются между ними.

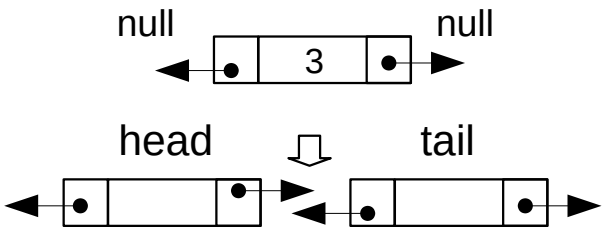
Схематическое изображение двусвязного списка



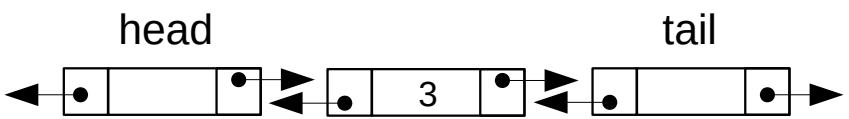
Добавление значения в начало списка



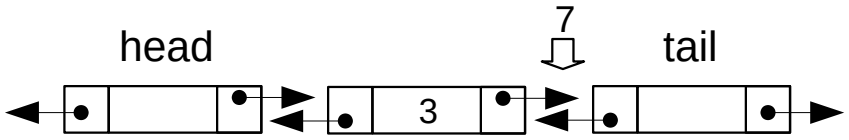
Создать новый узел который хранит добавляемое значение и next = null и prev = null



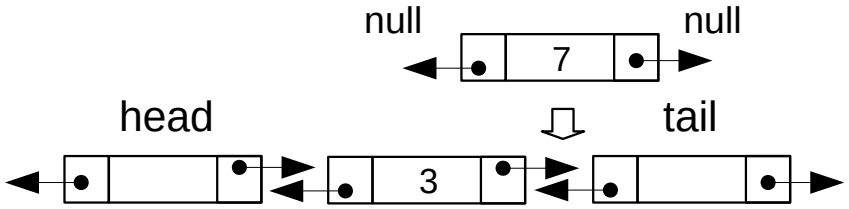
1) Установить значение next добавляемого узла равное head.next.
2) Установить prev добавляемого узла равное head.
3) Установить head.next.prev на добавляемый узел.
4) Установить head.next на добавляемый узел.



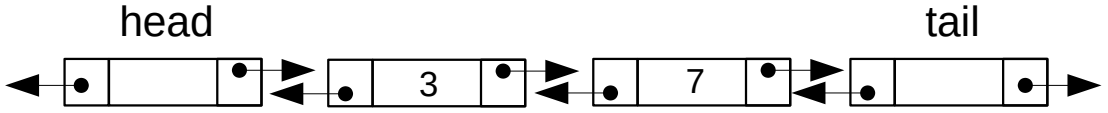
Добавление значения в конец списка



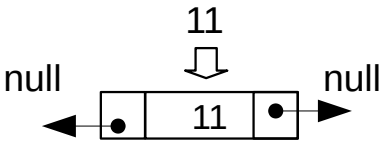
Создать новый узел который хранит добавляемое значение и next = null и prev = null



1) Установить значение next добавляемого узла равным tail.
2) Установить значение prev равный значению tail.prev.
3) Установить значение tail.prev.next равное добавляемому узлу.
4) Установить значение tail.prev равным добавляемому узлу.

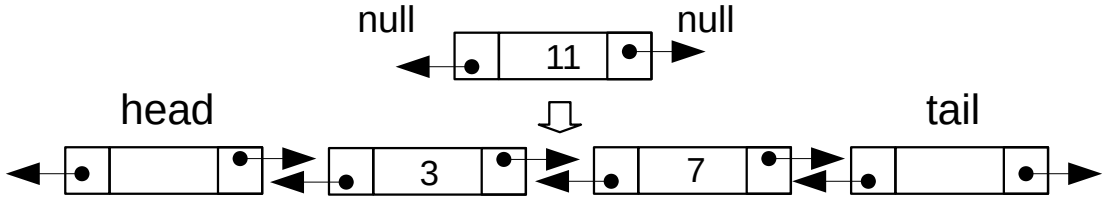


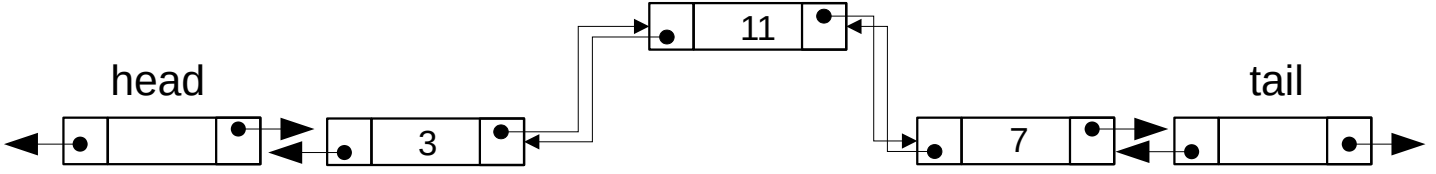
Добавление значения в произвольное место списка



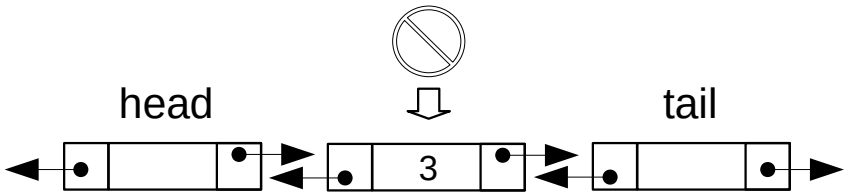
Создаем новый узел который хранит добавляемое значение и next = null, prev = null

1) Начиная с головы или хвоста списка проходим до элемента перед которым нужно вставить добавляемый узел.
2) Устанавливаем значение next добавляемого узла на текущий элемент.
3) Устанавливаем значение prev добавляемого узла равной значению prev текущего.
4) Устанавливаем next предыдущего узла равным добавляемому
5) Ссылку prev текущего равный добавляемому

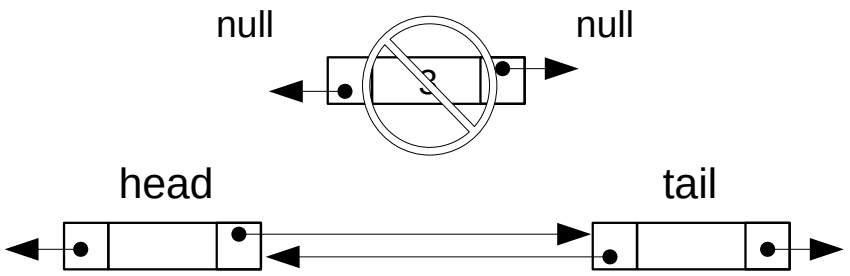




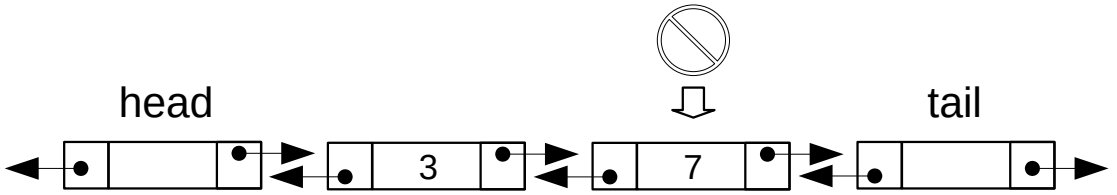
Удаление значения из начала списка



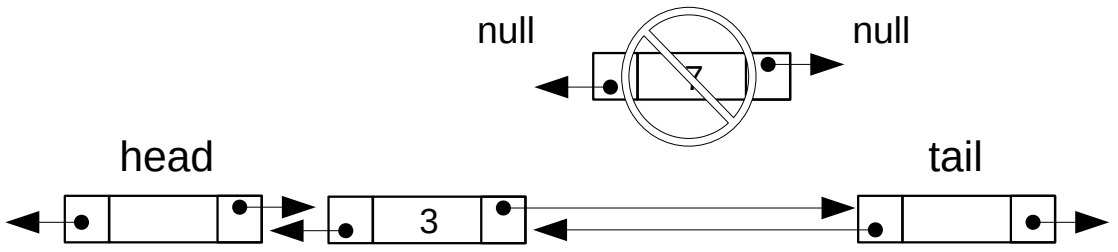
1) Установить значение head.next равное значению next удаляемого
2) Установить значение prev следующего элемента равным значению prev удаляемого
3) Указать, значение next и prev удаляемого узла равным null.
4) Освободить память занимаемую удаляемым узлом.



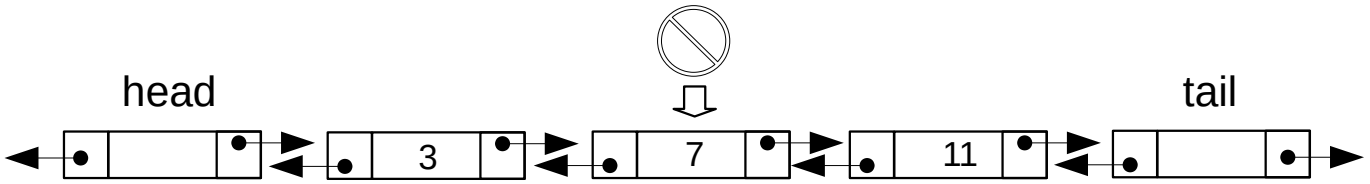
Удаление элемента из конца списка



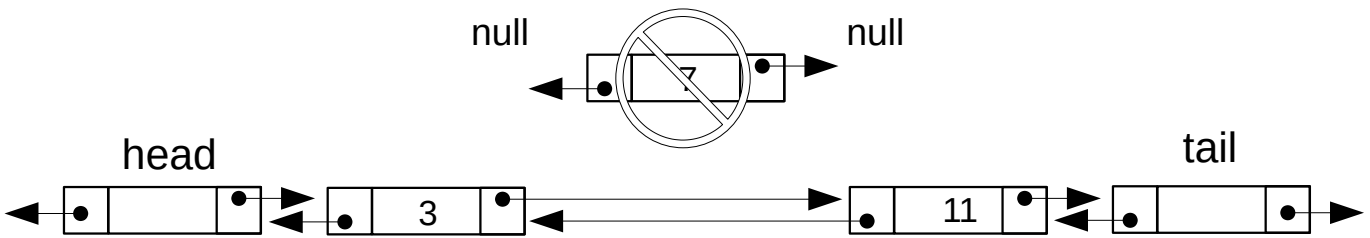
1) Устанавливаем значение tail.prev равным значению prev удаляемого узла
2) Устанавливаем значение next предыдущего узла равным значению tail
3) Устанавливаем значение next и prev удаляемого узла равными null
4) Освобождаем память занимаемую удаляемым узлом



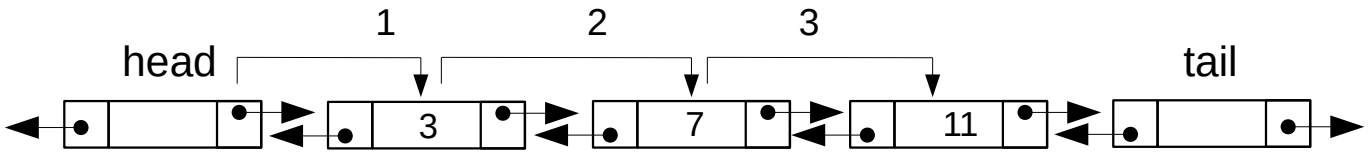
Удаление элемента из произвольного места списка



1) Начиная с головы или хвоста списка проходим до элемента который нужно удалить.
2) Устанавливаем значение next предыдущего узла равным next удаляемого.
3) Устанавливаем значение prev следующего узла равной значению prev удаляемого.
4) Устанавливаем next и prev удаляемого узла равным null.
5) Освобождаем память занимаемую удаляемым узлом.

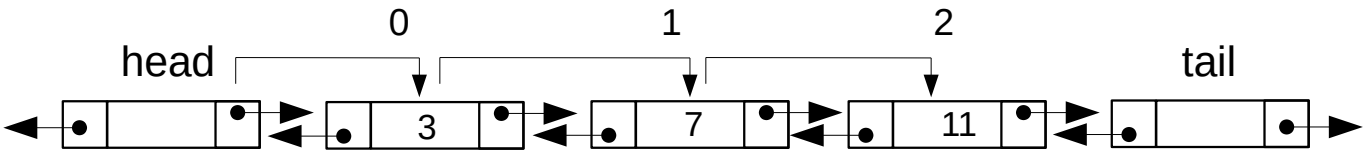


Получение размера списка



Для получения размера списка стоит объявить переменную с начальным значением 0. Начиная с начала списка выполнять переход по ссылке к следующему узлу. На каждом переходе увеличивать значение этой переменной на 1. Закончить на tail.

Работа с индексами



Для работы с индексами можно использовать подход аналогичный вычислению длины. Стоит объявить переменную начальное значение которой равно начальному индексу (произвольный выбор). Начиная с головы списка выполняем проход по next (от узла к узлу), то тех пор пока значение этой переменной не станет равно искомому индексу.

Python

Реализация на Python

Описание структуры узла и списка

```
class Node:
    def __init__(self, data=None, next=None, prev = None):
        self.data = data
        self.next = next
        self.prev = prev

    def __str__(self):
        return str(self.date)


class DoublyLinkedList:

    def __init__(self):
        self.head = Node()
        self.tail = Node()
        self.head.next = self.tail
        self.tail.prev = self.head
        self.length = 0
```

Методы добавления

```
def add_first(self,value):
    add_node = Node(value, self.head.next, self.head)
    self.head.next.prev = add_node
    self.head.next = add_node
    self.length += 1

def add_last(self, value):
    add_node = Node(value, self.tail, self.tail.prev)
    self.tail.prev.next = add_node
    self.tail.prev = add_node
    self.length += 1
```

Методы удаления

```
def remove_first(self):
    if self.is_empty():
        return
    remove_node = self.head.next
    self.head.next = remove_node.next
    remove_node.next.prev = self.head
    self.length -= 1


def remove_last(self):
    if self.is_empty():
        return
    remove_node = self.tail.prev
    self.tail.prev = remove_node.prev
    remove_node.prev.next = remove_node.next
    self.length -= 1

def remove_by_value(self, value):
    if self.is_empty():
        return
    current_node = self.head.next
    while current_node!=self.tail and current_node.data != value:
        current_node = current_node.next
    if current_node == self.tail:
        return
    current_node.prev.next = current_node.next
    current_node.next.prev = current_node.prev
    self.length -= 1
```

Методы для поиска узла по индексу

```
def get_node_by_index(self, index):
    if index < 0 or index >= self.length:
        raise IndexError()
    if index > self.length // 2:
        number = self.length - 1
        current_node = self.tail.prev
        while number != index:
            current_node = current_node.prev
            number -=1
    else:
        number = 0
        current_node = self.head.next
        while number != index:
            current_node = current_node.next
            number +=1
    return current_node
```

Методы для работы с индексами

```
def get_by_index(self, index):
    if index < 0 or index >= self.length:
        raise IndexError()
    return self.get_node_by_index(index).data

def remove_by_index(self, index):
    if index < 0 or index >= self.length:
        raise IndexError()
    remove_node = self.get_node_by_index(index)
    remove_node.next.prev = remove_node.prev
    remove_node.prev.next = remove_node.next
    remove_node.prev = None
    remove_node.next = None
    self.length -= 1

def insert_by_index(self, index, value):
    if index < 0 or index >= self.length:
        raise IndexError()
    current_node = self.get_node_by_index(index)
    add_node = Node(value, current_node, current_node.prev)
    current_node.prev.next = add_node
    current_node.prev = add_node
    self.length += 1
```

Методы для работы с индексами

```
def set_by_index(self, index, value):
    if index < 0 or index >= self.length:
        raise IndexError()
    current_node = self.get_node_by_index(index)
    current_node.data = value
```

Методы для получения размера

```
def get_length(self):
    current_node = self.head.next
    length = 0
    while current_node != self.tail:
        length += 1
        current_node = current_node.next
    return length
```

Реализация на Java

Описание узла

```
private class Node {
	Object data;
	Node next;
	Node prev;

	public Node(Object data, Node next, Node prev) {
		super();
		this.data = data;
		this.next = next;
		this.prev = prev;
	}

	public Node() {
		super();
	}
	@Override
	public String toString() {
		return "Node [data=" + data + ", next=" + next + ", prev=" + prev + "]";
	}
}
```

Класс двусвязный список

```
public class DoublyLinkedList {
     private final Node head;
     private final Node tail;
     private long length = 0;

     public DoublyLinkedList() {
          super();
          head = new Node();
          tail = new Node();
          head.next = tail;
          tail.prev = head;
     }
```

Методы добавления

```
public void addFirst(Object value) {
     Node addNode = new Node(value, head.next, head);
     head.next.prev = addNode;
     head.next = addNode;
     length += 1;
}

public void addLast(Object value) {
     Node addNode = new Node(value, tail, tail.prev);
     tail.prev.next = addNode;
     tail.prev = addNode;
     length += 1;
}
```

Методы удаления

```
public void removeFirst() {
     if (isEmpty()) {
          return;
     }
     Node removeNode = head.next;
     head.next = removeNode.next;
     removeNode.next.prev = head;
     removeNode.next = null;
     removeNode.prev = null;
     length -= 1;
}

public void removeLast() {
     if (isEmpty()) {
          return;
     }
     Node removeNode = tail.prev;
     tail.prev = removeNode.prev;
     removeNode.prev.next = tail;
     removeNode.next = null;
     removeNode.prev = null;
     length -= 1;
}
```

Метод для получения узла по индексу

```
private Node getNodeByIndex(long index) {
     Node resultNode = null;
     if (index < length / 2) {
          long nodeIndex = 0;
          resultNode = head.next;
          for (; nodeIndex != index;) {
               resultNode = resultNode.next;
               nodeIndex += 1;
          }
     } else {
          long nodeIndex = length - 1;
          resultNode = tail.prev;
          for (; nodeIndex != index;) {
               resultNode = resultNode.prev;
               nodeIndex -= 1;
          }
     }
     return resultNode;
}
```

Методы для работы с индексами

```
public Object getByIndex(long index) {
     if (index < 0 || index >= length) {
          throw new IndexOutOfBoundsException();
     }
     Node resultNode = getNodeByIndex(index);
     return resultNode.data;
}

public void removeByIndex(long index) {
     if (index < 0 || index >= length) {
          throw new IndexOutOfBoundsException();
     }
     Node resultNode = getNodeByIndex(index);
     resultNode.prev.next = resultNode.next;
     resultNode.next.prev = resultNode.prev;
     resultNode.next = null;
     resultNode.prev = null;
     length -= 1;
}
```

Методы для работы с индексами

```
public void insertByIndex(long index, Object value) {
     if (index < 0 || index >= length) {
          throw new IndexOutOfBoundsException();
     }
     Node resultNode = getNodeByIndex(index);
     Node addNode = new Node(value, resultNode, resultNode.prev);
     resultNode.prev.next = addNode;
     resultNode.prev = addNode;
     length += 1;
}
```

Метод для получения размера

```
public long getLength() {
     long length = 0;
     Node currentNode = head.next;
     for (; currentNode != tail;) {
          length += 1;
          currentNode = currentNode.next;
     }
     return length;
}
```

Реализация на Fortran

Описание узла и списка

```
type Node
    integer::data_value
    class(Node),pointer::next=>null()
    class(Node),pointer::prev=>null()
end type Node

type List
    class(Node),pointer::head => null()
    class(Node),pointer::tail => null()
    integer::length = 0
    contains
        procedure,pass::init
        procedure,pass::add_first
        procedure,pass::add_last
        procedure,pass::delete_first
        procedure,pass::delete_last
        procedure,pass::get_length
        procedure,pass::get_node_by_index
        procedure,pass::insert_by_index
        procedure,pass::delete_by_index
        procedure,pass::get_by_index
        procedure,pass::set_by_index
        procedure,pass::show_list
        procedure,pass::clear
        procedure,pass::destroy


end type List
```

Методы добавления

```
subroutine add_first(this, data_value)
    class(List)::this
    integer, intent(in)::data_value
    class (Node), pointer::new_node
    allocate(new_node)
    new_node%data_value = data_value
    new_node%next => this%head%next
    new_node%prev => this%head
    new_node%next%prev => new_node
    this%head%next => new_node
    this%length = this%length + 1
end subroutine add_first

subroutine add_last(this, data_value)
    class(List)::this
    integer, intent(in)::data_value
    class (Node), pointer::new_node
    allocate(new_node)
    new_node%data_value = data_value
    new_node%next => this%tail
    new_node%prev => this%tail%prev
    new_node%prev%next => new_node
    this%tail%prev => new_node
    this%length = this%length + 1
end subroutine add_last
```

Методы удаления

```
subroutine delete_first(this)
    class(List)::this
    class (Node), pointer::delete_node
    if(associated(this%head%next, this%tail)) then
        return
    end if
    delete_node => this%head%next
    this%head%next => delete_node%next
    delete_node%next%prev => delete_node%prev
    delete_node%next => null()
    delete_node%prev => null()
    deallocate(delete_node)
    this%length = this%length - 1
end subroutine delete_first

subroutine delete_last(this)
    class(List)::this
    class (Node), pointer::delete_node
    if(associated(this%head%next, this%tail)) then
        return
    end if
    delete_node => this%tail%prev
    this%tail%prev => delete_node%prev
    delete_node%prev%next => this%tail
    delete_node%next => null()
    delete_node%prev => null()
    deallocate(delete_node)
    this%length = this%length - 1
end subroutine delete_last
```

Метод для получения узла по индексу

```
subroutine get_node_by_index(this, node_index, result_node)
    class(List)::this
    integer, intent(in)::node_index
    class (Node), pointer, intent(inout)::result_node
    integer::current_index

    if (node_index < 1 .or. node_index > this%length) then
        return
    end if

    if (node_index < this%length / 2) then
        current_index = 1
        result_node => this%head%next
        do
            if(current_index == node_index) then
                exit
            end if
            current_index = current_index + 1
            result_node => result_node%next
        end do
     else
        current_index = this%length
        result_node => this%tail%prev
        do
            if(current_index == node_index) then
                exit
            end if
            current_index = current_index - 1
            result_node => result_node%prev
        end do
     end if
end subroutine get_node_by_index
```

Методы для работы с индексами

```
function get_by_index(this, node_index, op_result)
    class(List)::this
    integer, intent(in):: node_index
    logical, intent(inout)::op_result
    integer::get_by_index
    class(Node), pointer::current_node
    op_result = .false.
    current_node => null()
    call this%get_node_by_index(node_index, current_node)
    if(associated(current_node)) then
        get_by_index = current_node%data_value
        op_result = .true.
    end if
end function get_by_index

subroutine delete_by_index(this, node_index, op_result)
    class(List)::this
    integer, intent(in):: node_index
    logical, intent(inout)::op_result
    class(Node), pointer::current_node
    op_result = .false.
    current_node => null()
    call this%get_node_by_index(node_index, current_node)
    if(associated(current_node)) then
        current_node%prev%next => current_node%next
        current_node%next%prev => current_node%prev
        current_node%next => null()
        current_node%prev => null()
        deallocate(current_node)
        this%length = this%length - 1
        op_result = .true.
    end if
end subroutine delete_by_index
```

Методы для работы с индексами

```
subroutine set_by_index(this, node_index, data_value, op_result)
    class(List)::this
    integer, intent(in):: node_index
    logical, intent(inout)::op_result
    integer, intent(in)::data_value
    class(Node), pointer::current_node
    op_result = .false.
    current_node => null()
    call this%get_node_by_index(node_index, current_node)
    if(associated(current_node)) then
        current_node%data_value = data_value
        op_result = .true.
    end if
end subroutine set_by_index

subroutine insert_by_index(this, node_index, data_value, op_result)
    class(List)::this
    integer, intent(in):: node_index
    logical, intent(inout)::op_result
    integer, intent(in)::data_value
    class(Node), pointer::current_node
    class(Node), pointer::new_node
    op_result = .false.
    current_node => null()
    call this%get_node_by_index(node_index, current_node)
    if(associated(current_node)) then
        allocate(new_node)
        new_node%data_value = data_value
        new_node%next => current_node
        new_node%prev => current_node%prev
        current_node%prev%next => new_node
        current_node%prev =>new_node
        this%length = this%length + 1
        op_result = .true.
    end if
end subroutine insert_by_index
```

Метод для получения длины

```
function get_length(this)
    class(List)::this
    class(Node), pointer::current_node
    integer::get_length
    get_length = 0
    current_node => this%head%next
    do
        if(associated(current_node,this%tail)) then
            exit
        end if
        get_length = get_length + 1
        current_node => current_node%next
    end do
end function get_length
```

Метод для создания, очистки, и удаления списка

```
subroutine init(this)
    class(List)::this
    allocate(this%head)
    allocate(this%tail)
    this%head%next => this%tail
    this%tail%prev => this%head
end subroutine init

subroutine clear(this)
    class(List)::this
    do
        if(this%length == 0) then
            exit
        end if
        call this%delete_first()
    end do
end subroutine clear

subroutine destroy(this)
    class(List)::this
    if( .not. associated(this%head)) then
        return
    end if
    if(this%length > 0) then
        call this%clear()
    end if
    this%head%prev => null()
    this%head%next => null()
    this%tail%prev => null()
    this%tail%next => null()
    deallocate(this%head)
    deallocate(this%tail)
end subroutine destroy
```

Список литературы

1) Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн // Алгоритмы: построение и анализ 3-е издание. — М.: «Вильямс», 2013. — С. 1328. ISBN 978-5-8459-1794-2
2) Роберт Седжвик, Кевин Уэйн «Алгоритмы на java 4-е издание» Пер. с англ. - М. : ООО "И.Д. Вильямс", 2013. ISBN 978-5-8459-1781-2.