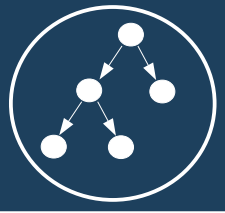


Data Structures and Algorithms

Структуры данных. Односвязный список

Связанный список

Связанный список - базовая динамическая структура данных (одна из возможных реализаций списка), состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или две ссылки(указатели) на следующий и/или предыдущий узел списка.

Линейный односвязный список (односвязный список) — разновидность связанного списка. Узел содержит данные и одну ссылку (указатель) на следующий элемент списка. Довольно часто используется также альтернативное определение. Односвязный список — рекурсивная линейная структура данных, которая либо пуста, либо ссылается на узел (который хранит данные и ссылку на следующий узел).

Преимущества и недостатки связанных списков

Как уже было сказано выше связанные списки это динамическая структура данных. По сравнению с массивами она обладает рядом как преимуществ так и недостатков.

Преимущества:

- Возможность добавления и удаления элементов (изменение размера списка)
- Размер ограничен доступной оперативной памятью

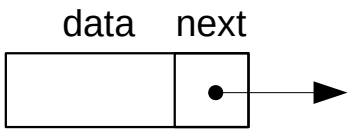
Недостатки:

- Сложность получения элемента по индексу
- Дополнительный расход памяти на хранение указателей
- Нелокальное хранение данных списка (снижает вероятность кеширования)
- Сложность в выполнении параллельной обработки

Узел списка

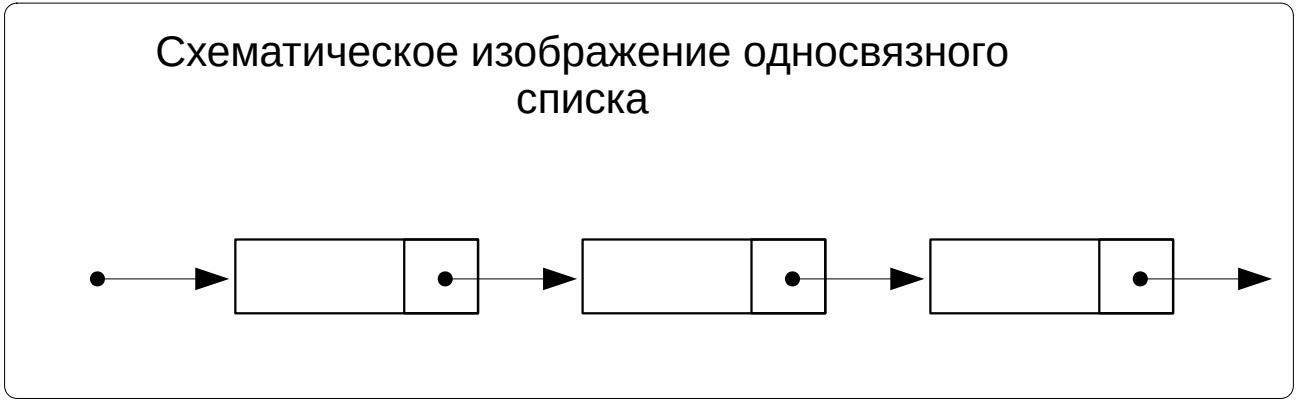
Узел списка представляет собой составную структуру. Обычно реализуется с помощью класса или структуры (в процедурных языках). Содержит значения двух типов. Одно для хранения данных (числа, строки и т. д.), второе это ссылка на следующий узел (реализуется как указать или ссылка тип которых совпадает с типом узел). Из за того, что на момент компиляции количество узлов списка не известно, то эти память для их хранения должна выделяться динамически (т. е. использовать heap).

Схематическое изображение узла списка

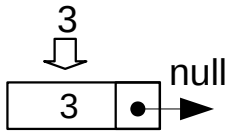


Односвязный список

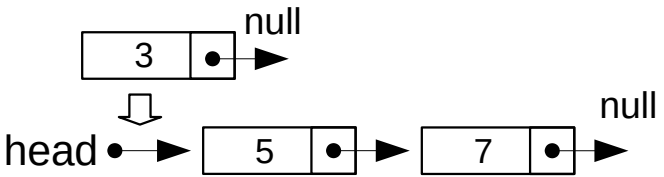
В простейшем случае односвязный список состоит из структуры хранящей одну ссылку на узел. Такая ссылка называется головой списка (head). Существует два подхода относительно головы списка. В первом случае эта ссылка не инициализирована (равна null, None и т. д.), во втором подходе она указывает на фиктивный узел (создается при инициализации списка). При добавлении значений в список, голова списка указывает на следующий узел в списке (который указывает на следующий узел и т. д.). Такая структура дополняется рядом методов (функций) для добавления, удаления, получения узлов.



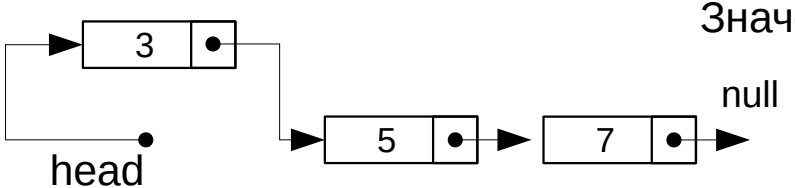
Добавление значения в начало списка

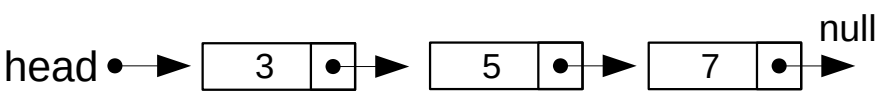


При добавлении нового значения в список, нужно создать новый узел который хранит добавляемое значение и next = null

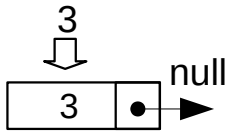


Установить значение next добавляемого узла равное head. Значение head установить равным ссылке на добавляемый узел.

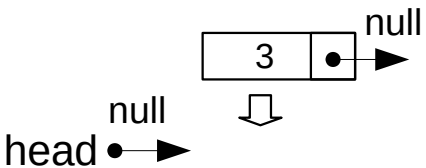




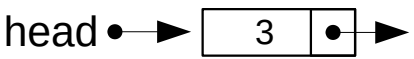
Добавление значения в конец списка. Список пуст



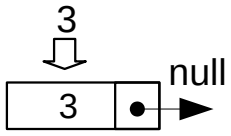
При добавлении нового значения в список, нужно создать новый узел который хранит добавляемое значение и next = null



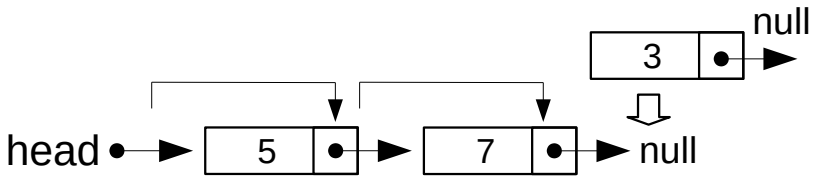
Значение head установить равным ссылке на добавляемый узел



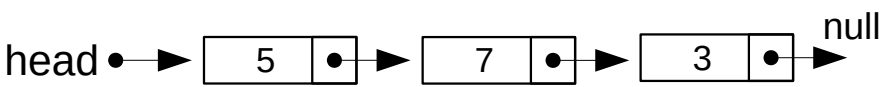
Добавление значения в конец списка. Список не пуст



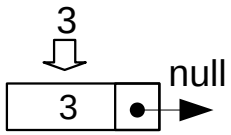
При добавлении нового значения в список, нужно создать новый узел который хранит добавляемое значение и next = null.



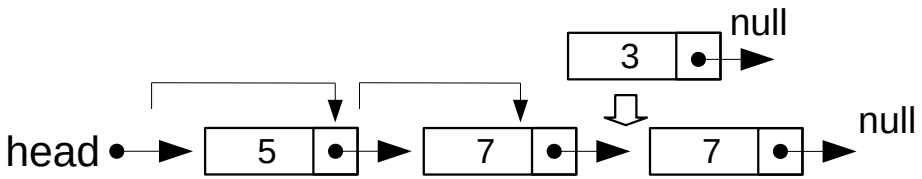
Начиная с начала списка проверяем next узла. Если next != null то переходим к следующему узлу. В случае next == null (текущий узел является последним), то устанавливаем ее на добавляемый узел.



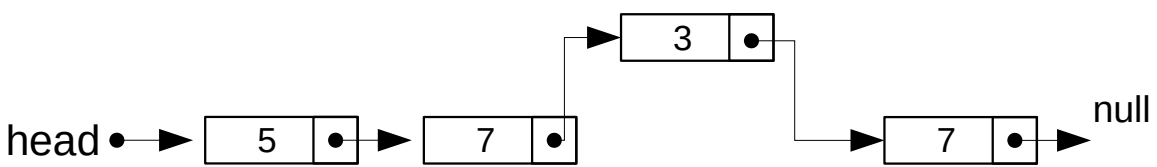
Добавление значения в произвольное место списка



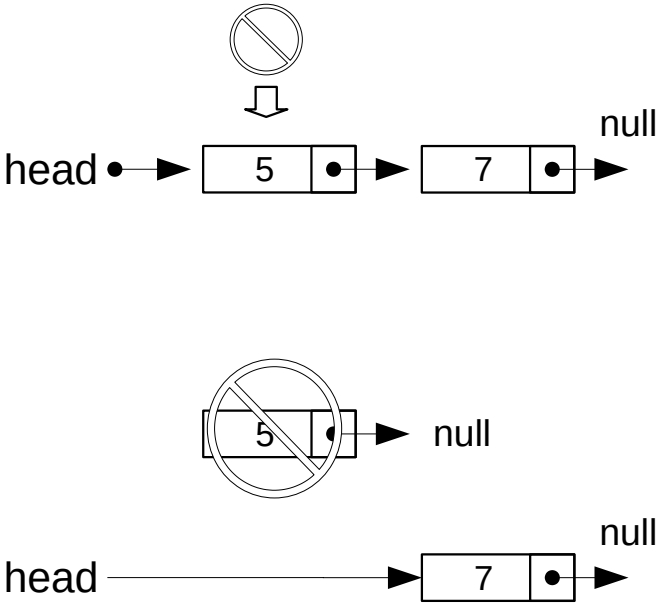
Создаем новый узел который хранит добавляемое значение и next = null



Начиная с головы списка проходим до элемента после которого нужно вставить добавляемый узел. Устанавливаем значение next добавляемого узла, значение равное next текущего. Ссылку next текущего устанавливаем на добавляемый элемент.

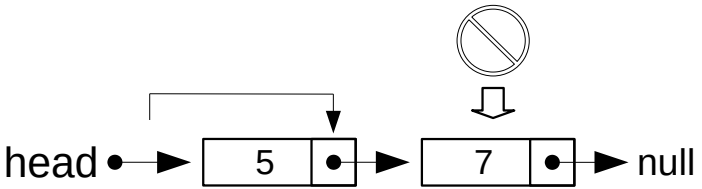


Удаление значения из начала списка

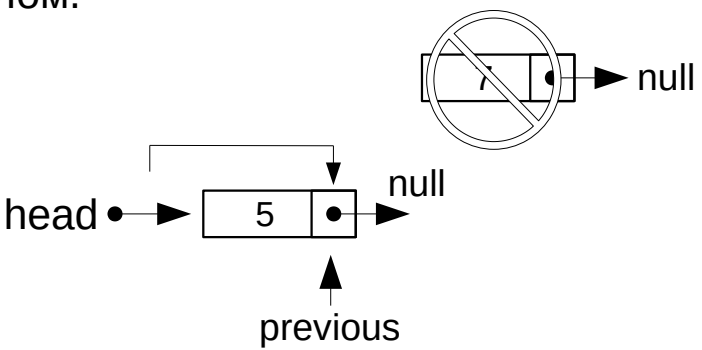


Установить значение head равное значению next удаляемого элемента. Указать, значение next удаляемого узла равным null. Освободить память занимаемую удаляемым узлом.

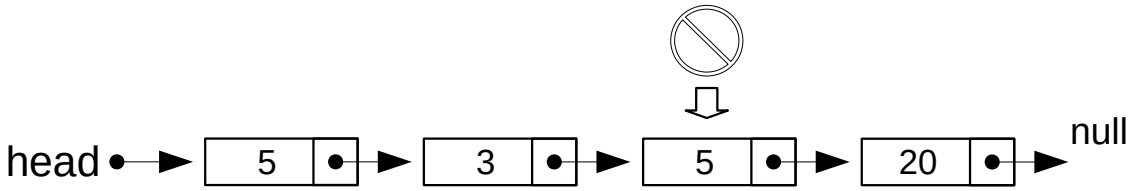
Удаление элемента из конца списка



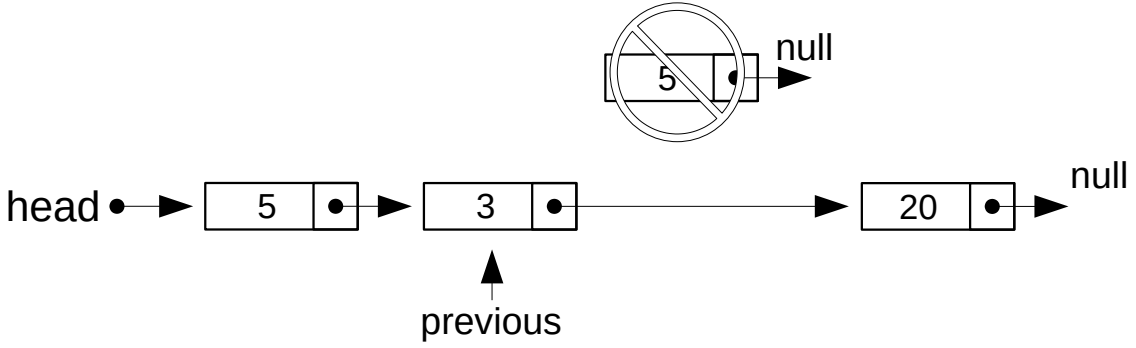
Используем дополнительную ссылку (previous). Начиная с головы списка выполняем проход по узлам, причем в ссылке previos сохраняем ссылку на предыдущий узел. Как только next текущего узла станет равна null, то переходим к узлу по ссылке previos (это предпоследний элемент) устанавливаем значение next для него равным null. Освобождаем память занимаемую удаляемым узлом.



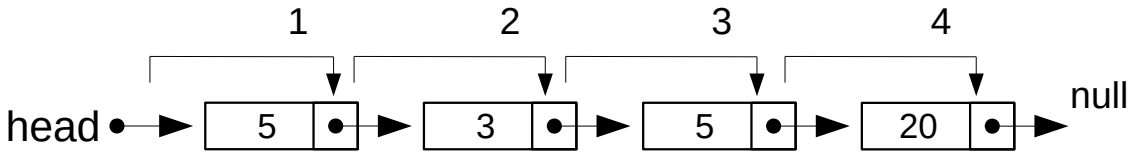
Удаление элемента из произвольного места списка



Используем дополнительную ссылку (previous). Начиная с головы списка выполняем проход по узлам, причем в ссылке previos сохраняем ссылку на предыдущий узел. Как только текущей узел станет равен удаляемому, то переходим к узлу по ссылке previos устанавливаем значение next для него равным значению next удаляемого узла. Устанавливаем значение next = null для удаляемого. Освобождаем память занимаемую удаляемым узлом.

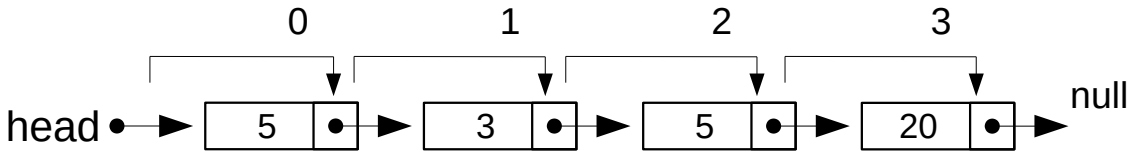


Получение размера списка



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 узле для которого next == null.

Работа с индексами



Для работы с индексами можно использовать подход аналогичный вычислению длинны. 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 узла,списка и методы добавления

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

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


class LinkedList:

    def __init__(self):
        self.head = None

    def add_first(self, value):
        new_node = Node(value)
        new_node.next = self.head
        self.head = new_node

    def add_last(self, value):
        new_node = Node(value)
        if self.head is None:
            self.head = new_node
            return
        current_node = self.head
        while current_node.next is not None:
            current_node = current_node.next
        current_node.next = new_node
```

Методы удаления

```
def delete_last(self):
     if self.head is None or self.head.next is None:
          self.head = None
          return
     current_node = self.head
     previous_node = current_node
     while current_node.next is not None:
          previous_node = current_node
          current_node = current_node.next
     previous_node.next = None

def delete_first(self):
     if self.head is None:
          return
     self.head = self.head.next
```

Методы для работы с индексами

```
def insert_by_index(self, value, index):
      if index == 0:
            self.add_first(value)
            return
      new_node = Node(value)
      node_number = 1
      current_node = self.head
      while current_node is not None:
            if index == node_number:
                  new_node.next = current_node.next
                  current_node.next = new_node
                  return
            current_node = current_node.next
            node_number += 1

def delete_by_index(self, index):
      if index == 0:
            self.delete_first()
            return
      node_number = 0
      current_node = self.head
      previous_node = current_node
      while current_node is not None:
            if index == node_number:
                  previous_node.next = current_node.next
                  current_node.next = None
                  return
            previous_node = current_node
            current_node = current_node.next
            node_number += 1
```

Методы для работы с индексами

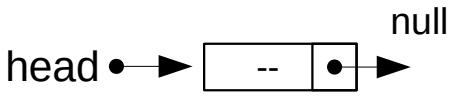
```
def set_value_by_index(self, value, index):
      node_number = 0
      current_node = self.head
      while current_node is not None:
            if node_number == index:
                  current_node.data = value
                  return
            current_node = current_node.next
            node_number += 1

def get_value_by_index(self, index):
      node_number = 0
      current_node = self.head
      while current_node is not None:
            if index == node_number:
                  return current_node.data
            current_node = current_node.next
            node_number += 1
      return None
```

Методы для получения размера

```
def get_length(self):
      length = 0
      current_node = self.head
      while current_node is not None:
            length += 1
            current_node = current_node.next
      return length
```

Использование фиктивного начального узла

Для упрощения реализации односвязного списка может быть использован фиктивный первый узел. Голова списка всегда указывает на фиктивный узел. Данные этого узла игнорируются. Применение фиктивного узла позволяет уменьшить количество проверок при реализации. К недостаткам стоит отнести необходимость выделения памяти для хранения фиктивного узла.



Реализация на Java

Описание узла и списка

```
class LinkedList {
     private class Node {
          int date;
          Node next;

          public Node(int date, Node next) {
               this.date = date;
               this.next = next;
          }

          public Node() {
          }

     }

     private Node head;

     public LinkedList() {
          head = new Node();
     }
```

Методы добавления

```
public void addFirst(int value) {
     Node newNode = new Node(value, head.next);
     head.next = newNode;
}

public void addLast(int value) {
     Node currentNode = head;
          for (; currentNode.next != null; currentNode = currentNode.next) {

     }
     currentNode.next = new Node(value, null);
}
```

Методы удаления

```
public void deleteFirst() {
     head.next = (head.next == null) ? null : head.next.next;
}

public void deleteLast() {
     if (head.next == null) {
          return;
     }
     Node curretNode = head.next;
     Node previousNode = head;
     for (; curretNode.next != null; curretNode = curretNode.next) {
          previousNode = curretNode;
     }
     previousNode.next = null;
}
```

Методы для работы с индексами

```
public void insertByIndex(int value, long index) {
	long nodeNumber = 0;
	for (Node currentNode = head; currentNode != null; currentNode = currentNode.next) {
		if (nodeNumber == index) {
			currentNode.next = new Node(value, currentNode.next);
			return;
		}
		nodeNumber += 1L;
	}
	throw new IndexOutOfBoundsException();
}

public void deleteByIndex(long index) {
	long nodeNumber = 0;
	Node curretNode = head.next;
	Node previousNode = head;
	for (; curretNode != null; curretNode = curretNode.next) {
		if (nodeNumber == index) {
			previousNode.next = curretNode.next;
			curretNode.next = null;
			return;
		}
		previousNode = curretNode;
		nodeNumber += 1;
	}
	throw new IndexOutOfBoundsException();
}
```

Методы для работы с индексами

```
public int getByIndex(long index) {
     long nodeNumber = 0;
     Node curretNode = head.next;
     for (; curretNode != null; curretNode = curretNode.next) {
          if (nodeNumber == index) {
               return curretNode.date;
          }
          nodeNumber += 1L;
     }
     throw new IndexOutOfBoundsException();
}

public void setByIndex(int value, long index) {
     long nodeNumber = 0;
     Node curretNode = head.next;
     for (; curretNode != null; curretNode = curretNode.next) {
          if (nodeNumber == index) {
               curretNode.date = value;
               return;
          }
          nodeNumber += 1L;
     }
     throw new IndexOutOfBoundsException();
}
```

Методы для получения размера

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

Реализация на Fortran

Описание узла и списка

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

type List
    class(Node), pointer::head=>null()

    contains
        procedure,pass::add_first
        procedure,pass::add_last
        procedure,pass::delete_first
        procedure,pass::delete_last
        procedure,pass::get_length
        procedure,pass::insert_by_index
        procedure,pass::delete_by_index
        procedure,pass::get_by_index
        procedure,pass::set_by_index
        procedure,pass::clear
        procedure,pass::show_list
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
    this%head => new_node
end subroutine add_first

subroutine add_last(this, data_value)
    class(List)::this
    integer, intent(in)::data_value
    class (Node), pointer::new_node
    class(Node), pointer::current_node
    if(.not. associated (this%head)) then
        call this%add_first(data_value)
        return
    end if
    allocate(new_node)
    new_node%data_value = data_value
    current_node => this%head
    do
        if(.not. associated (current_node%next)) then
            exit
        end if
        current_node => current_node%next
    end do
    current_node%next => new_node
end subroutine add_last
```

Методы удаления

```
    subroutine delete_first(this)
        class(List)::this
        class(Node),pointer::current_node
        if(.not. associated (this%head)) then
            return
        end if
        current_node => this%head%next
        this%head%next => null()
        deallocate (this%head)
        this%head => current_node
    end subroutine delete_first

    subroutine delete_last(this)
        class(List)::this
        class(Node),pointer::current_node
        class(Node),pointer::previous_node
        if(.not. associated (this%head)) then
            return
        end if
        current_node => this%head%next
        if(.not. associated (current_node)) then
            call this%delete_first()
            return
        end if
        previous_node => this%head
        do
            if(.not. associated (current_node%next)) then
                exit
            end if
            previous_node => current_node
            current_node => current_node%next
        end do
        previous_node%next => null()
        deallocate(current_node)
    end subroutine delete_last
```

Методы для работы с индексами

```
subroutine insert_by_index(this, data_value, indx, res)
        class(List)::this
        integer,intent(in)::data_value
        integer(8),intent(in)::indx
        logical,intent(inout)::res
        class(Node), pointer::current_node, previous_node
        class(Node), pointer::new_node
        integer(8)::node_index = 1
        res = .false.
        if(indx == 1) then
            call this%add_first(data_value)
            res = .true.
            return
        end if
        node_index = 2
        previous_node => this%head
        current_node => this%head%next
        do
            if(.not. associated (current_node)) then
                return
            end if
            if(node_index == indx) then
                allocate(new_node)
                new_node%data_value = data_value
                new_node%next => current_node
                previous_node%next => new_node
                res = .true.
                return
            end if
            node_index=node_index + 1_8
            previous_node => current_node
            current_node => current_node%next
        end do
    end subroutine insert_by_index
```

Методы для работы с индексами

```
    subroutine delete_by_index(this, indx, res)
        class(List)::this
        integer(8),intent(in)::indx
        logical,intent(inout)::res
        class(Node), pointer::current_node, previous_node
        integer(8)::node_index = 2
        res = .false.
        if (indx == 1) then
            call this%delete_first()
            res = .true.
            return
        end if
        previous_node => this%head
        current_node => this%head%next
        do
            if(.not. associated (current_node)) then
                return
            end if
            if(node_index == indx) then
                previous_node%next => current_node%next
                current_node%next => null()
                deallocate(current_node)
                res = .true.
                return
            end if
            node_index=node_index + 1_8
            previous_node => current_node
            current_node => current_node%next
        end do
    end subroutine delete_by_index
```

Методы для работы с индексами

```
integer function get_by_index(this, indx, res)
    class(List)::this
    integer(8),intent(in)::indx
    logical,intent(inout)::res
    class(Node), pointer::current_node
    integer(8)::node_index = 1
    current_node=> this%head
    do
        if(.not. associated (current_node)) then
            return
        end if
        if(node_index == indx) then
            get_by_index = current_node%data_value
            res = .true.
            return
        end if
        node_index=node_index + 1_8
        current_node => current_node%next
    end do
end function get_by_index
```

Методы для работы с индексами

```
subroutine set_by_index(this, data_value, indx, res)
    class(List)::this
    integer(8),intent(in)::indx
    integer,intent(in)::data_value
    logical,intent(inout)::res
    class(Node), pointer::current_node
    integer(8)::node_index = 1
    current_node=> this%head
    do
        if(.not. associated (current_node)) then
            return
        end if
        if(node_index == indx) then
            current_node%data_value = data_value
            res = .true.
            return
        end if
        node_index=node_index + 1_8
        current_node => current_node%next
    end do
end subroutine set_by_index
```

Метод для получения длины

```
integer(8) function get_length(this)
    class(List)::this
    integer(8)::list_size = 0
    class(Node),pointer::current_node
    current_node => this%head
    do
        if(.not. associated (current_node)) then
            exit
        end if
        list_size=list_size + 1_8
        current_node => current_node%next
    end do
    get_length = list_size
end function get_length
```

Список литературы

1) 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.