



Обозначим n – номер варианта. Функция добавления элемента в список выбирается учащимся исходя из значения n % 4+1

1 вариант



1. fun push (el: T): Bool вставляет элемент в начало списка;

class ListNode<T>(var value: T, var next: ListNode<T>? = null)  
  
class LinkedList<T> {  
 var head: ListNode<T>? = null  
  
 fun push(el: T) {  
 val newNode = ListNode(el)  
 newNode.next = head  
 head = newNode  
 }  
  
 fun display() {  
 var current = head  
 while (current != null) {  
 *print*("${current.value} -> ")  
 current = current.next  
 }  
 *println*("null")  
 }  
}  
  
fun main() {  
 val linkedList = LinkedList<Int>()  
  
 linkedList.push(3)  
 linkedList.push(7)  
 linkedList.push(11)  
  
 linkedList.display()  
}

(+)

Процедура удаления элемента из списка выбирается учащимся исходя из значения n/4%4+1:

сделать остальные

2. fun delete(): Bool удаляет элемент с конца списка

class ListNode<T>(var value: T, var next: ListNode<T>? = null)  
  
class LinkedList<T> {  
 var head: ListNode<T>? = null  
  
 fun push(el: T) {  
 val newNode = ListNode(el)  
 newNode.next = head  
 head = newNode  
 }  
  
 fun display() {  
 var current = head  
 while (current != null) {  
 *print*("${current.value} -> ")  
 current = current.next  
 }  
 *println*("null")  
 }  
  
 fun delete(): Boolean {  
 if (head == null) {  
 return false // Список пуст, нельзя удалить элемент  
 } else if (head?.next == null) {  
 head = null // В списке только один элемент, удаляем его  
 return true  
 }  
  
 var prev: ListNode<T>? = null  
 var current = head  
  
 while (current?.next != null) {  
 prev = current  
 current = current.next  
 }  
  
 prev?.next = null // Удаляем последний элемент  
 return true  
 }  
}  
  
fun main() {  
 val linkedList = LinkedList<Int>()  
  
 linkedList.push(3)  
 linkedList.push(7)  
 linkedList.push(11)  
  
 linkedList.display()  
  
 val deleted = linkedList.delete()  
 if (deleted) {  
 *println*("Последний элемент удален успешно")  
 linkedList.display()  
 } else {  
 *println*("Невозможно удалить последний элемент")  
 }  
}

Процедура печати элементов списка выбирается учащимся исходя из значения n mod 5+1:

2. fun print():Unit печатает первый элемент списка

class ListNode<T>(var value: T, var next: ListNode<T>? = null)  
  
class LinkedList<T> {  
 var head: ListNode<T>? = null  
  
 fun push(el: T) {  
 val newNode = ListNode(el)  
 newNode.next = head  
 head = newNode  
 }  
  
 fun display() {  
 var current = head  
 while (current != null) {  
 *print*("${current.value} -> ")  
 current = current.next  
 }  
 *println*("null")  
 }  
  
 fun print() {  
 if (head == null) {  
 *println*("Список пуст")  
 } else {  
 *println*("Первый элемент списка: ${head?.value}")  
 }  
 }  
  
 fun delete(): Boolean {  
 if (head == null) {  
 return false // Список пуст, нельзя удалить элемент  
 } else if (head?.next == null) {  
 head = null // В списке только один элемент, удаляем его  
 return true  
 }  
  
 var prev: ListNode<T>? = null  
 var current = head  
  
 while (current?.next != null) {  
 prev = current  
 current = current.next  
 }  
  
 prev?.next = null // Удаляем последний элемент  
 return true  
 }  
}  
  
fun main() {  
 val linkedList = LinkedList<Int>()  
  
 linkedList.push(3)  
 linkedList.push(7)  
 linkedList.push(11)  
  
 linkedList.display()  
  
 linkedList.print() // Новая функция для вывода первого элемента  
  
 val deleted = linkedList.delete()  
 if (deleted) {  
 *println*("Последний элемент удален успешно")  
 linkedList.display()  
 } else {  
 *println*("Невозможно удалить последний элемент")  
 }  
}

class ListNode<T>(var value: T, var next: ListNode<T>? = null)  
  
class LinkedList<T> {  
 var head: ListNode<T>? = null  
  
 fun push(el: T) {  
 val newNode = ListNode(el)  
 newNode.next = head  
 head = newNode  
 }  
  
 fun display() {  
 var current = head  
 while (current != null) {  
 *print*("${current.value} -> ")  
 current = current.next  
 }  
 *println*("null")  
 }  
  
 fun print() {  
 if (head == null) {  
 *println*("Список пуст")  
 } else {  
 *println*("Первый элемент списка: ${head?.value}")  
 }  
 }  
  
 fun delete(): Boolean {  
 if (head == null) {  
 return false // Список пуст, нельзя удалить элемент  
 } else if (head?.next == null) {  
 head = null // В списке только один элемент, удаляем его  
 return true  
 }  
  
 var prev: ListNode<T>? = null  
 var current = head  
  
 while (current?.next != null) {  
 prev = current  
 current = current.next  
 }  
  
 prev?.next = null // Удаляем последний элемент  
 return true  
 }  
  
 fun eraseAll() {  
 head = null // Обнуляем голову списка, таким образом, очищая весь список  
 }  
}  
  
fun main() {  
 val linkedList = LinkedList<Int>()  
  
 linkedList.push(3)  
 linkedList.push(7)  
 linkedList.push(11)  
  
 linkedList.display()  
  
 linkedList.print() // Новая функция для вывода первого элемента  
  
 linkedList.eraseAll() // Очищаем весь список  
  
 *println*("Список после удаления:")  
 linkedList.display()  
}