МИНИСТРЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОУ НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ИМ. Р.Е. АЛЕКСЕЕВА

ИНСТИТУТ ЭКОНОМИКИ И УПРАВЛЕНИЯ

КАФЕДРА " Прикладная математика и информатика"

Дисциплина "Структуры данных"

Отчет

по лабораторной работе № 5

Выполнил: Иванов Михаил Егорович

студент группы: 21 САИ

Проверил: Моисеев Антон Евгеньевич

Нижний Новгород

2023

**Цель работы:** реализовать связный список.

**Практическая часть.**

Связный список - это структура данных, состоящая из узлов, каждый из которых содержит данные и указатель на следующий узел в списке. Связный список позволяет эффективно добавлять и удалять элементы в произвольных позициях, но требует дополнительной памяти для хранения указателей и не поддерживает произвольный доступ к элементам.

Для реализации связного списка была создана структура Node, которая представляет элемент списка, и структура List, которая представляет сам список. Структура Node содержит поле data для хранения данных типа int и поле next для хранения указателя на следующий узел типа \*Node. Структура List содержит поле head для хранения указателя на головной узел списка типа \*Node.

Для работы со связным списком были созданы следующие методы:

* Метод append для добавления нового узла в конец списка. Метод принимает данные для нового узла и создает его с помощью оператора &. Затем метод проверяет, пуст ли список, и если да, то делает новый узел головным. Иначе метод ищет последний узел в списке и связывает его с новым узлом.
* Метод insert для добавления нового узла по заданному индексу. Метод принимает данные для нового узла и индекс, по которому он должен быть вставлен. Затем метод проверяет, что индекс не отрицательный и не превышает длину списка. Если индекс равен нулю, то метод делает новый узел головным и связывает его со старым головным. Иначе метод ищет узел, который предшествует заданному индексу, и связывает новый узел с ним и с узлом по заданному индексу.
* Метод remove для удаления узла по заданному индексу. Метод принимает индекс удаляемого узла и проверяет, что он не отрицательный и не превышает длину списка. Если индекс равен нулю, то метод удаляет головной узел и делает следующий узел головным. Иначе метод ищет узел, который предшествует заданному индексу, и связывает его с узлом, который следует за удаляемым узлом.
* Метод length для подсчета длины списка. Метод начинает с головного узла и счетчика, равного нулю, и пока не достигнет конца списка, увеличивает счетчик на единицу и переходит к следующему узлу. Затем метод возвращает счетчик в качестве длины списка.
* Метод print для вывода элементов списка на экран. Метод начинает с головного узла и пока не достигнет конца списка, выводит данные каждого узла на экран.

**Код:**

package main  
  
import "fmt"  
  
// Node Определяем структуру Node, которая представляет элемент связного списка  
type Node struct {  
 data int // Данные, хранящиеся в узле  
 next \*Node // Указатель на следующий узел в списке  
}  
  
// List Определяем структуру List, которая представляет связный список  
type List struct {  
 head \*Node // Указатель на головной узел списка  
}  
  
// Создаем метод append для добавления нового узла в конец списка  
func (l \*List) append(data int) {  
 // Создаем новый узел с данными  
 newNode := &Node{data: data}  
 // Если список пуст, то делаем новый узел головным  
 if l.head == nil {  
 l.head = newNode  
 return  
 }  
 // Иначе ищем последний узел в списке  
 last := l.head  
 for last.next != nil {  
 last = last.next  
 }  
 // Добавляем новый узел в конец списка  
 last.next = newNode  
}  
  
// Создаем метод insert для добавления нового узла по заданному индексу  
func (l \*List) insert(data int, index int) {  
 // Проверяем, что индекс не отрицательный и не превышает длину списка  
 if index < 0 || index > l.length() {  
 fmt.Println("Неверный индекс")  
 return  
 }  
 // Создаем новый узел с данными  
 newNode := &Node{data: data}  
 // Если индекс равен нулю, то делаем новый узел головным и связываем его со старым головным  
 if index == 0 {  
 newNode.next = l.head  
 l.head = newNode  
 return  
 }  
 // Иначе ищем узел, который предшествует заданному индексу  
 prev := l.head  
 for i := 0; i < index-1; i++ {  
 prev = prev.next  
 }  
 // Связываем новый узел с узлом по заданному индексу и с предыдущим узлом  
 newNode.next = prev.next  
 prev.next = newNode  
}  
  
// Создаем метод remove для удаления узла по заданному индексу  
func (l \*List) remove(index int) {  
 // Проверяем, что индекс не отрицательный и не превышает длину списка  
 if index < 0 || index >= l.length() {  
 fmt.Println("Неверный индекс")  
 return  
 }  
 // Если индекс равен нулю, то удаляем головной узел и делаем следующий узел головным  
 if index == 0 {  
 l.head = l.head.next  
 return  
 }  
 // Иначе ищем узел, который предшествует заданному индексу  
 prev := l.head  
 for i := 0; i < index-1; i++ {  
 prev = prev.next  
 }  
 // Связываем предыдущий узел с узлом, который следует за удаляемым узлом  
 prev.next = prev.next.next  
}  
  
// Создаем метод length для подсчета длины списка  
func (l \*List) length() int {  
 // Начинаем с головного узла и счетчика, равного нулю  
 current := l.head  
 count := 0  
 // Пока не достигнем конца списка, увеличиваем счетчик на единицу и переходим к следующему узлу  
 for current != nil {  
 count++  
 current = current.next  
 }  
 // Возвращаем счетчик в качестве длины списка  
 return count  
}  
  
// Создаем метод print для вывода элементов списка на экран  
func (l \*List) print() {  
 // Начинаем с головного узла  
 current := l.head  
 // Пока не достигнем конца списка, выводим данные каждого узла  
 for current != nil {  
 fmt.Println(current.data)  
 current = current.next  
 }  
}  
  
func main() {  
 // Создаем пустой список  
 list := &List{}  
 // Добавляем несколько элементов в список  
 list.append(1)  
 list.append(2)  
 list.append(3)  
 list.append(4)  
 list.append(5)  
 // Выводим список на экран  
 fmt.Println("Список после добавления элементов:")  
 list.print()  
 // Добавляем элемент по индексу 2  
 list.insert(6, 2)  
 // Выводим список на экран  
 fmt.Println("Список после вставки элемента по индексу 2:")  
 list.print()  
 // Удаляем элемент по индексу 4  
 list.remove(4)  
 // Выводим список на экран  
 fmt.Println("Список после удаления элемента по индексу 4:")  
 list.print()  
}

### Вывод: