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 № 9

Выполнил: Иванов Михаил Егорович

студент группы: 21 САИ

Проверил: Моисеев Антон Евгеньевич

Нижний Новгород

2023

**Цель работы:** реализовать стек поверх массива.

**Практическая часть.**

Стек - это структура данных, в которой элементы добавляются и удаляются из одного конца, то есть работает по принципу LIFO (Last In First Out). Массив - это структура данных, в которой элементы хранятся в непрерывной области памяти и имеют произвольный доступ по индексу.

Для реализации стека на основе массива была создана структура Stack, которая представляет стек, и содержит поле data для хранения массива, который хранит элементы стека. Массив имеет динамический размер и может быть расширен или сокращен с помощью функции append из пакета builtin.

Для работы с стеком на основе массива были созданы следующие методы:

* Метод push для добавления элемента в вершину стека. Метод принимает данные для нового элемента и добавляет их в конец массива, который представляет стек, с помощью функции append.
* Метод pop для удаления элемента из вершины стека и возвращения его данных. Метод проверяет, что стек не пуст, и если нет, то запоминает данные последнего элемента массива, который представляет стек, удаляет его из массива, уменьшая его длину на единицу, и возвращает его данные.
* Метод size для определения размера стека. Метод возвращает длину массива, который представляет стек.
* Метод print для вывода элементов стека на экран. Метод выводит элементы массива, который представляет стек, в обратном порядке.

**Код:**

package main  
  
import "fmt"  
  
// Определяем структуру Stack, которая представляет стек на основе массива  
type Stack struct {  
 data []int // Массив, который хранит элементы стека  
}  
  
// Создаем метод push для добавления элемента в вершину стека  
func (s \*Stack) push(data int) {  
 // Добавляем элемент в конец массива, который представляет стек  
 s.data = append(s.data, data)  
}  
  
// Создаем метод pop для удаления элемента из вершины стека и возвращения его данных  
func (s \*Stack) pop() int {  
 // Проверяем, что стек не пуст  
 if len(s.data) == 0 {  
 fmt.Println("Стек пуст")  
 return 0  
 }  
 // Запоминаем данные последнего элемента массива, который представляет стек  
 data := s.data[len(s.data)-1]  
 // Удаляем последний элемент массива, уменьшая его длину на единицу  
 s.data = s.data[:len(s.data)-1]  
 // Возвращаем данные удаленного элемента  
 return data  
}  
  
// Создаем метод size для определения размера стека  
func (s \*Stack) size() int {  
 // Возвращаем длину массива, который представляет стек  
 return len(s.data)  
}  
  
// Создаем метод print для вывода элементов стека на экран  
func (s \*Stack) print() {  
 // Выводим элементы массива, который представляет стек, в обратном порядке  
 for i := len(s.data) - 1; i >= 0; i-- {  
 fmt.Println(s.data[i])  
 }  
}  
  
func main() {  
 // Создаем пустой стек на основе пустого массива  
 stack := &Stack{data: []int{}}  
 // Добавляем несколько элементов в стек  
 stack.push(1)  
 stack.push(2)  
 stack.push(3)  
 stack.push(4)  
 stack.push(5)  
 // Выводим стек на экран  
 fmt.Println("Стек после добавления элементов:")  
 stack.print()  
 // Удаляем элемент из стека и выводим его данные  
 fmt.Println("Удаленный элемент из стека:")  
 fmt.Println(stack.pop())  
 // Выводим стек на экран  
 fmt.Println("Стек после удаления элемента:")  
 stack.print()  
}

### Вывод: