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 № 7

Выполнил: Иванов Михаил Егорович

студент группы: 21 САИ

Проверил: Моисеев Антон Евгеньевич

Нижний Новгород

2023

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

**Практическая часть.**

Очередь - это структура данных, в которой элементы добавляются в конец и удаляются из начала, то есть работает по принципу FIFO (First In First Out). Массив - это структура данных, в которой элементы хранятся в непрерывной области памяти и имеют произвольный доступ по индексу.

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

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

* Метод enqueue для добавления элемента в конец очереди. Метод принимает данные для нового элемента и добавляет их в конец массива, который представляет очередь, с помощью функции append.
* Метод dequeue для удаления элемента из начала очереди и возвращения его данных. Метод проверяет, что очередь не пуста, и если нет, то запоминает данные первого элемента массива, который представляет очередь, удаляет его из массива, сдвигая остальные элементы на одну позицию влево, и возвращает его данные.
* Метод insert для добавления элемента по заданному индексу. Метод принимает данные для нового элемента и индекс, по которому он должен быть вставлен. Затем метод проверяет, что индекс не отрицательный и не превышает размер очереди. Если это так, то метод создает новый массив с дополнительной позицией для нового элемента, копирует элементы массива до заданного индекса в новый массив, вставляет новый элемент в новый массив по заданному индексу и копирует оставшиеся элементы массива в новый массив после заданного индекса. Затем метод заменяет старый массив новым.
* Метод remove для удаления элемента по заданному индексу. Метод принимает индекс удаляемого элемента и проверяет, что он не отрицательный и не превышает размер очереди. Если это так, то метод создает новый массив с меньшей позицией для удаленного элемента, копирует элементы массива до заданного индекса в новый массив, копирует оставшиеся элементы массива в новый массив после заданного индекса, пропуская удаленный элемент. Затем метод заменяет старый массив новым.
* Метод size для определения размера очереди. Метод возвращает длину массива, который представляет очередь.
* Метод print для вывода элементов очереди на экран. Метод выводит элементы массива, который представляет очередь.

**Код:**

package main  
  
import "fmt"  
  
// Определяем структуру Queue, которая представляет очередь на основе массива  
type Queue struct {  
 data []int // Массив, который хранит элементы очереди  
}  
  
// Создаем метод enqueue для добавления элемента в конец очереди  
func (q \*Queue) enqueue(data int) {  
 // Добавляем элемент в конец массива, который представляет очередь  
 q.data = append(q.data, data)  
}  
  
// Создаем метод dequeue для удаления элемента из начала очереди и возвращения его данных  
func (q \*Queue) dequeue() int {  
 // Проверяем, что очередь не пуста  
 if len(q.data) == 0 {  
 fmt.Println("Очередь пуста")  
 return 0  
 }  
 // Запоминаем данные первого элемента массива, который представляет очередь  
 data := q.data[0]  
 // Удаляем первый элемент массива, сдвигая остальные элементы на одну позицию влево  
 q.data = q.data[1:]  
 // Возвращаем данные удаленного элемента  
 return data  
}  
  
// Создаем метод insert для добавления элемента по заданному индексу  
func (q \*Queue) insert(data int, index int) {  
 // Проверяем, что индекс не отрицательный и не превышает размер очереди  
 if index < 0 || index > len(q.data) {  
 fmt.Println("Неверный индекс")  
 return  
 }  
 // Создаем новый массив с дополнительной позицией для нового элемента  
 newData := make([]int, len(q.data)+1)  
 // Копируем элементы массива до заданного индекса в новый массив  
 copy(newData[:index], q.data[:index])  
 // Вставляем новый элемент в новый массив по заданному индексу  
 newData[index] = data  
 // Копируем оставшиеся элементы массива в новый массив после заданного индекса  
 copy(newData[index+1:], q.data[index:])  
 // Заменяем старый массив новым  
 q.data = newData  
}  
  
// Создаем метод remove для удаления элемента по заданному индексу  
func (q \*Queue) remove(index int) {  
 // Проверяем, что индекс не отрицательный и не превышает размер очереди  
 if index < 0 || index >= len(q.data) {  
 fmt.Println("Неверный индекс")  
 return  
 }  
 // Создаем новый массив с меньшей позицией для удаленного элемента  
 newData := make([]int, len(q.data)-1)  
 // Копируем элементы массива до заданного индекса в новый массив  
 copy(newData[:index], q.data[:index])  
 // Копируем оставшиеся элементы массива в новый массив после заданного индекса, пропуская удаленный элемент  
 copy(newData[index:], q.data[index+1:])  
 // Заменяем старый массив новым  
 q.data = newData  
}  
  
// Создаем метод size для определения размера очереди  
func (q \*Queue) size() int {  
 // Возвращаем длину массива, который представляет очередь  
 return len(q.data)  
}  
  
// Создаем метод print для вывода элементов очереди на экран  
func (q \*Queue) print() {  
 // Выводим элементы массива, который представляет очередь  
 for \_, data := range q.data {  
 fmt.Println(data)  
 }  
}  
  
func main() {  
 // Создаем пустую очередь на основе пустого массива  
 queue := &Queue{data: []int{}}  
 // Добавляем несколько элементов в очередь  
 queue.enqueue(1)  
 queue.enqueue(2)  
 queue.enqueue(3)  
 queue.enqueue(4)  
 queue.enqueue(5)  
 // Выводим очередь на экран  
 fmt.Println("Очередь после добавления элементов:")  
 queue.print()  
 // Удаляем элемент из очереди и выводим его данные  
 fmt.Println("Удаленный элемент из очереди:")  
 fmt.Println(queue.dequeue())  
 // Выводим очередь на экран  
 fmt.Println("Очередь после удаления элемента:")  
 queue.print()  
 // Добавляем элемент по индексу 2  
 queue.insert(6, 2)  
 // Выводим очередь на экран  
 fmt.Println("Очередь после вставки элемента по индексу 2:")  
 queue.print()  
 // Удаляем элемент по индексу 4  
 queue.remove(4)  
 // Выводим очередь на экран  
 fmt.Println("Очередь после удаления элемента по индексу 4:")  
 queue.print()  
}

### Вывод: