Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Курский государственный университет»

Кафедра программного обеспечения и администрирования информационных систем

Направление подготовки математическое обеспечение и администрирование информационных систем

Форма обучения очная

**Отчет**

**по лабораторной работе №5**

«Составление процедур, реализующих операции с очередью»

Выполнил:

студент группы 213 Файтельсон А.А.

Проверил:

ассистент кафедры ПОиАИС Овсянников А.В.

Курск, 2025

**Цель работы:** Научиться составлять процедуры, реализующие операции:

* 1. создание очереди;
  2. помещение звена с заданным информационным элементом в очередь;
  3. удаление звена из очереди.

**Словесная постановка задачи**

Необходимо реализовать структуру данных "очередь" и выполнить следующие операции:

1. Создание очереди.
2. Добавление элемента в очередь.
3. Удаление элемента из очереди.

На основе реализованной очереди необходимо решить две задачи:

1. Переписать содержимое текстового файла F в текстовый файл G, перенося при этом в конец каждой строки все входящие в нее цифры (с сохранением исходного взаимного порядка как среди цифр, так и среди остальных символов строки).
2. Для заданной очереди целых чисел:
   1. Вычислить сумму и произведение всех элементов.
   2. Подсчитать количество простых чисел в очереди.

**Алгоритм решения задачи в текстуальном виде**

**Задача 1**: Обработка текстового файла

1. Открыть файл F.txt для чтения.
2. Создать файл G.txt для записи результатов.
3. Для каждой строки в файле F.txt:
   1. Разделить символы на цифры и остальные символы, используя две очереди:
      1. digitsQueue для хранения цифр.
      2. otherCharsQueue для хранения остальных символов.
   2. Собрать новую строку, добавляя сначала символы из otherCharsQueue, затем — из digitsQueue.
   3. Записать новую строку в файл G.txt.
4. Закрыть файлы F.txt и G.txt.

**Задача 2**: Работа с очередью целых чисел

1. Создать очередь queue и заполнить ее заданными целыми числами.
2. Инициализировать переменные для суммы (sum), произведения (product) и счетчика простых чисел (primeCount).
   1. Пройти по всем элементам очереди:
   2. Прибавить текущий элемент к sum.
   3. Умножить текущий элемент на product.
   4. Проверить, является ли текущий элемент простым числом, и увеличить primeCount при необходимости.
3. Вывести значения sum, product и primeCount.

**Обоснование правильности выбора алгоритма**

Для решения задач использовалась структура данных "очередь", которая обеспечивает FIFO-порядок обработки элементов. Это позволяет эффективно выполнять операции добавления и удаления элементов.

Преимущества выбранного подхода:

Задача 1 : Использование двух очередей (digitsQueue и otherCharsQueue) позволяет легко разделить символы на цифры и остальные символы, сохраняя их порядок.

Задача 2 : Проход по очереди с использованием метода Values() обеспечивает удобную итерацию по элементам без изменения самой очереди.

**Листинг программы**

package main

import (

"bufio"

"fmt"

"math"

"os"

"unicode"

)

// LinkedListQueueExample to demonstrate basic usage of LinkedListQueue

//

// func main() {

// queue := llq.New() // empty

// queue.Enqueue(1) // 1

// queue.Enqueue(2) // 1, 2

// \_ = queue.Values() // 1, 2 (FIFO order)

// \_, \_ = queue.Peek() // 1,true

// \_, \_ = queue.Dequeue() // 1, true

// \_, \_ = queue.Dequeue() // 2, true

// \_, \_ = queue.Dequeue() // nil, false (nothing to deque)

// queue.Enqueue(1) // 1

// queue.Clear() // empty

// queue.Empty() // true

// \_ = queue.Size() // 0

// queue.de

// }

// Очередь с использованием generics

type Queue[T any] struct {

data []T

}

// Создание новой очереди

func NewQueue[T any]() \*Queue[T] {

return &Queue[T]{data: make([]T, 0)}

}

// Добавление элемента в очередь

func (q \*Queue[T]) Enqueue(value T) {

q.data = append(q.data, value)

}

// Удаление и возврат элемента из очереди

func (q \*Queue[T]) Dequeue() (T, bool) {

if len(q.data) == 0 {

var zero T // Нулевое значение для типа T

return zero, false

}

value := q.data[0]

q.data = q.data[1:] // Удаляем первый элемент

return value, true

}

// Проверка, пуста ли очередь

func (q \*Queue[T]) IsEmpty() bool {

return len(q.data) == 0

}

// Получение длины очереди

func (q \*Queue[T]) Len() int {

return len(q.data)

}

// Метод String для получения строкового представления очереди

func (q \*Queue[T]) String() string {

return fmt.Sprint(q.data)

}

// Метод Values для получения всех элементов очереди в виде среза

func (q \*Queue[T]) Values() []T {

return q.data

}

func main() {

// Открываем файл F для чтения

fileF, err := os.Open("F.txt")

if err != nil {

fmt.Println("Ошибка при открытии файла F:", err)

return

}

defer fileF.Close()

// Создаем файл G для записи

fileG, err := os.Create("G.txt")

if err != nil {

fmt.Println("Ошибка при создании файла G:", err)

return

}

defer fileG.Close()

scanner := bufio.NewScanner(fileF)

writer := bufio.NewWriter(fileG)

// Обрабатываем каждую строку файла F

for scanner.Scan() {

line := scanner.Text()

digitsQueue := NewQueue[rune]() // Очередь для цифр

otherCharsQueue := NewQueue[rune]() // Очередь для остальных символов

// Разделяем символы на цифры и остальные

for \_, char := range line {

if unicode.IsDigit(char) {

digitsQueue.Enqueue(char)

} else {

otherCharsQueue.Enqueue(char)

}

}

// Собираем строку из символов, не являющихся цифрами

newLine := otherCharsQueue.String()

// Добавляем цифры в конец строки

newLine += digitsQueue.String()

// Записываем обработанную строку в файл G

\_, err := writer.WriteString(newLine + "\n")

if err != nil {

fmt.Println("Ошибка при записи в файл G:", err)

return

}

}

// Проверяем, были ли ошибки при сканировании файла F

if err := scanner.Err(); err != nil {

fmt.Println("Ошибка при чтении файла F:", err)

return

}

// Сбрасываем буфер записи в файл G

writer.Flush()

fmt.Println("Файл успешно обработан. Результат записан в G.txt")

// Функция для проверки, является ли число простым

// Создаем очередь и добавляем элементы

queue := NewQueue[int]()

data := []int{2, 3, 4, 5, 7, 10}

for \_, element := range data {

queue.Enqueue(element)

}

// Инициализация переменных для суммы, произведения и счетчика простых чисел

sum := 0

product := 1

primeCount := 0

// Проходим по очереди

for \_, element := range queue.Values() {

value := element

sum += value

product \*= value

if isPrime(value) {

primeCount++

}

}

// Выводим результаты

fmt.Printf("Сумма элементов очереди: %d\n", sum)

fmt.Printf("Произведение элементов очереди: %d\n", product)

fmt.Printf("Количество простых чисел в очереди: %d\n", primeCount)

}

func isPrime(n int) bool {

if n < 2 {

return false

}

for i := 2; i <= int(math.Sqrt(float64(n))); i++ {

if n%i == 0 {

return false

}

}

return true

}