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 №6**

«Составление процедур, реализующих операции с стеком»

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 стека;
  2. Помещение элемента в стек;
  3. Удаление элемента из стека;
  4. Разбиение типизированного файла с целыми числами на два файла согласно заданному условию.

**Словесная постановка задачи**

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

1. **Разбиение файла:**  Дан типизированный файл, элементами которого являются целые числа, упорядоченные по убыванию. Написать программу, которая разбивает файл на два типизированных файла, элементы которых упорядочены по возрастанию так, чтобы каждый элемент первого файла (a1) был больше соответствующего элемента второго файла (b1), но меньше его следующего элемента (b2), т.е. b1 < a1 < b2.
2. **Операция с добавлением элемента:**  Предположим, что уже построен стек, элементами которого являются целые числа. Написать программу, которая добавляет в конец стека элемент, находящийся в его вершине.

**Алгоритм решения задачи в текстуальном виде**

#### **Задача 1: Разбиение файла**

* 1. Открыть исходный файл для чтения.
  2. Создать два новых файла для записи результатов: fileA.txt и fileB.txt.
  3. Прочитать все числа из исходного файла, сохраняя их в массив.
  4. Инициализировать два стека: stackA и stackB.
  5. Проитерировать по числам из массива в обратном порядке:
     1. Если вершина стека stackB пуста или текущее число больше вершины stackB, поместить число в stackB.
     2. Иначе, если вершина стека stackA пуста или текущее число больше вершины stackA, поместить число в stackA.
  6. Преобразовать содержимое стеков в массивы, соблюдая порядок.
  7. Записать массивы в соответствующие файлы: числа из stackA в fileA.txt, числа из stackB в fileB.txt.
  8. Закрыть все файлы.

**Обоснование правильности выбора алгоритма**

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

Преимущества выбранного подхода:

* **Задача 1** : Использование двух стеков (stackA и stackB) позволяет легко разделить числа из исходного файла на две группы, соблюдая заданное условие. Проход по числам в обратном порядке гарантирует корректное разбиение.
* **Задача 2** : Метод AddTopToBottom() реализован с использованием базовых операций стека (Peek() и Push()), что делает код простым и понятным.

**Листинг программы**

package main

import (

"bufio"

"fmt"

"os"

"strconv"

"strings"

)

// Реализация стека

type Stack struct {

data []int

}

// Метод для добавления элемента в стек

func (s \*Stack) Push(value int) {

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

}

// Метод для удаления элемента из стека

func (s \*Stack) Pop() (int, bool) {

if len(s.data) == 0 {

return 0, false

}

top := s.data[len(s.data)-1]

s.data = s.data[:len(s.data)-1]

return top, true

}

// Метод для получения верхнего элемента стека без удаления

func (s \*Stack) Peek() (int, bool) {

if len(s.data) == 0 {

return 0, false

}

return s.data[len(s.data)-1], true

}

// Метод для добавления элемента из вершины стека в конец

func (s \*Stack) AddTopToBottom() {

if len(s.data) == 0 {

fmt.Println("Стек пуст.")

return

}

// Получаем верхний элемент

top, \_ := s.Peek()

// Добавляем его в конец стека

s.Push(top)

}

// Вывод содержимого стека

func (s \*Stack) Print() {

fmt.Println("Стек:", s.data)

}

// Функция для чтения чисел из текстового файла

func readNumbersFromFile(filename string) ([]int, error) {

file, err := os.Open(filename)

if err != nil {

return nil, err

}

defer file.Close()

var numbers []int

scanner := bufio.NewScanner(file)

for scanner.Scan() {

line := scanner.Text()

parts := strings.Fields(line)

for \_, part := range parts {

num, err := strconv.Atoi(part)

if err != nil {

continue // Пропускаем некорректные значения

}

numbers = append(numbers, num)

}

}

return numbers, scanner.Err()

}

// Функция для записи чисел в текстовый файл

func writeNumbersToFile(filename string, numbers []int) error {

file, err := os.Create(filename)

if err != nil {

return err

}

defer file.Close()

for \_, num := range numbers {

\_, err := fmt.Fprintln(file, num)

if err != nil {

return err

}

}

return nil

}

// Основная функция разбиения файла с использованием стека

func splitFile(inputFile, fileA, fileB string) error {

// Чтение исходного файла

numbers, err := readNumbersFromFile(inputFile)

if err != nil {

return err

}

// Инициализация двух стеков

stackA := Stack{}

stackB := Stack{}

// Разбиение чисел по условию b1 < a1 < b2

for i := len(numbers) - 1; i >= 0; i-- {

peekB, \_ := stackB.Peek()

peekA, \_ := stackA.Peek()

if peekB == 0 || numbers[i] > peekB {

stackB.Push(numbers[i])

} else if peekA == 0 || numbers[i] > peekA {

stackA.Push(numbers[i])

}

}

// Преобразование стеков в массивы

aNumbers := make([]int, len(stackA.data))

bNumbers := make([]int, len(stackB.data))

for i := len(stackA.data) - 1; i >= 0; i-- {

aNumbers[len(stackA.data)-1-i], \_ = stackA.Pop()

}

for i := len(stackB.data) - 1; i >= 0; i-- {

bNumbers[len(stackB.data)-1-i], \_ = stackB.Pop()

}

// Запись результатов в файлы

if err := writeNumbersToFile(fileA, aNumbers); err != nil {

return err

}

if err := writeNumbersToFile(fileB, bNumbers); err != nil {

return err

}

return nil

}

func main() {

inputFile := "input.txt"

fileA := "fileA.txt"

fileB := "fileB.txt"

err := splitFile(inputFile, fileA, fileB)

if err != nil {

fmt.Println("Ошибка:", err)

} else {

fmt.Println("Файл успешно разбит на два файла.")

}

stack := Stack{}

// Добавляем элементы в стек

stack.Push(1)

stack.Push(2)

stack.Push(3)

fmt.Println("Исходный стек:")

stack.Print()

// Добавляем элемент из вершины в конец

stack.AddTopToBottom()

fmt.Println("После добавления элемента из вершины в конец:")

stack.Print()

}