МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

Отчет по практической работе №2

по дисциплине «Тестирование и верификация ПО»

Выполнили:
Студенты группы ИКБО-15-22 Оганнисян Г.А.

Проверил:
Доцент Чернов Е.А.

2024 г.

Содержание

Техническое задание на разработку программы для работы со строками

1. Общие сведения

Данная программа написана на языке Go и предназначена для выполнения различных операций со строками. Программа позволяет пользователю вводить строку и выполнять над ней различные действия, такие как изменение регистра, разворот строки, удаление букв или цифр, и другие функции.

2. Цели и назначение создания автоматизированной системы

Цель программы — автоматизировать обработку строк, предоставив пользователю возможность выбора нескольких функций для модификации строки. Программа может использоваться в учебных целях для изучения строковых операций и взаимодействия с пользователем через консоль.

3. Характеристика объектов автоматизации

Объектом автоматизации является строка, введенная пользователем через консоль. Программа позволяет производить следующие операции со строкой:

- Разворот регистра символов.
- Обратный порядок символов строки.
- Разделение строки на части по заданному количеству символов.
- Удаление всех букв из строки.
- Удаление всех цифр из строки.

4. Требования к автоматизированной системе

Программа должна работать на любом компьютере с установленным компилятором для языка Go. Для запуска программы пользователь должен ввести строку, после чего ему предлагается выбрать одно из возможных действий. Программа должна завершаться корректно по запросу пользователя. Допустимый ввод — любая строка, включающая как буквы, так и цифры.

5. Состав и содержание работ по созданию автоматизированной системы

1. Написание кода программы.
2. Тестирование каждой функции на корректность работы.
3. Описание известных ошибок в коде.
4. Документирование программы.

6. Порядок разработки автоматизированной системы

1. Разработка структуры программы с использованием типа данных Str для представления строки.
2. Реализация функций для изменения строки.
3. Закладка намеренной ошибки в функции OnlyDigit, которая удваивает числа при их выводе.
4. Тестирование программы с рфазличными строками и анализ результатов.

7. Порядок контроля и приемки автоматизированной системы

Контроль программы осуществляется через выполнение различных сценариев:

- Проверка работы всех функций программы с корректным вводом.
- Валидация работы при вводе некорректных данных.
- Подтверждение того, что программа завершает работу при выборе команды завершения.

8. Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу автоматизированной системы в действие

Для использования программы пользователю необходимо ввести строку через консоль, а затем выбрать соответствующую операцию для обработки строки. В программе предусмотрена возможность ввода новой строки в любой момент.

9. Требования к документированию

Программа должна сопровождаться документацией, описывающей функциональные возможнофсти каждой функции:

- Flip(): изменяет регистр всех символов строки.
- Reverse(): переворачивает строку в обратном порядке.
- Del(n int): разбивает строку на части по n символов.
- OnlyDigit(): оставляет в строке только цифры (включает ошибку — дублирует цифры).
- OnlySim(): оставляет в строке только символы, отличные от цифр.

10. Источники разработки

- Язык программирования Go: Документация Go.
- Стандартные библиотеки Go для работы с консольным вводом и строками.

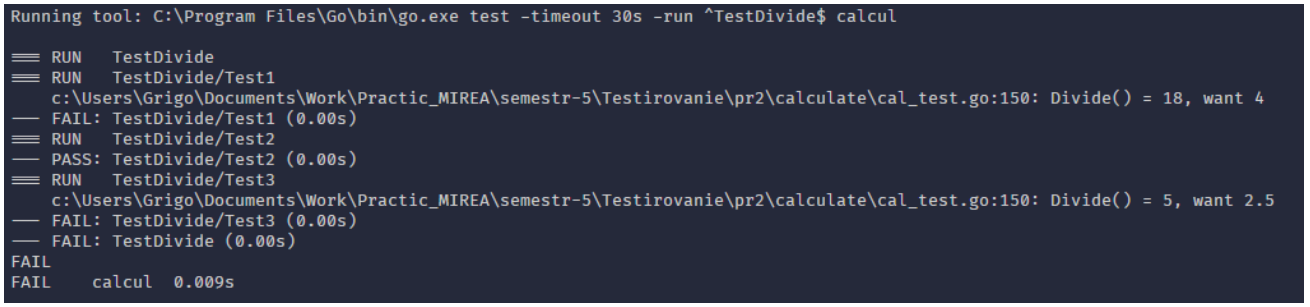

Рисунок 1 – Обнаруженная ошибка в модуле TestDivide.

Краткое описание ошибки: «Возвращает некорректный результат (умножение вместо деления и деление результата на 2)»

Статус ошибки: Открыта («Open»)

Категория ошибки: Серьезная («Major»)

Тестовый случай: «Проверка корректности работы функции деления»

Описание ошибки:

1. Загрузить программу.
2. Нажать кнопку «Запуск» программы.
3. Ввести значение двух чисел (12, 3)
4. Выбрать: /
5. Полученный результат: 18.
6. Ожидаемый результат: 4.

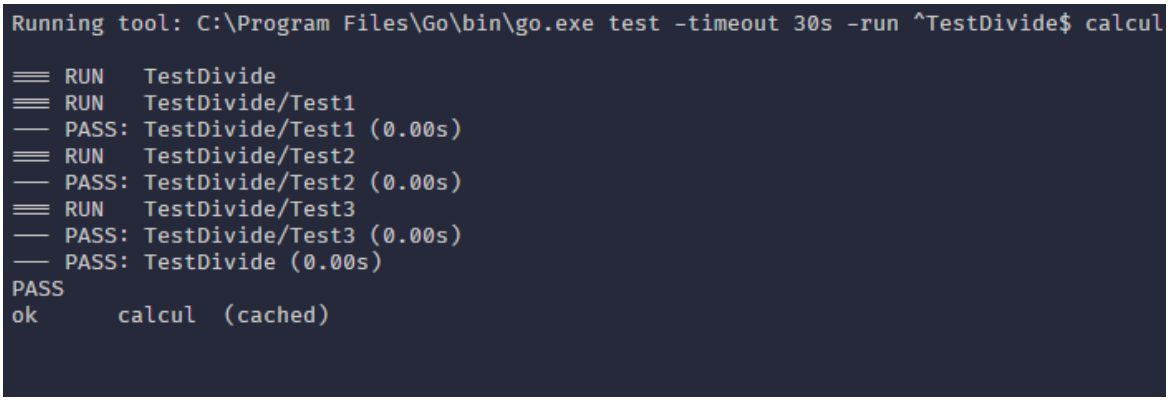

Рисунок 2 – Успешно пройденные Unit-тесты, после исправления ошибки создателем

ЗАКЛЮЧЕНИЕ

В рамках практической работы №2 были получены навыки писать Unit-тесты для поиска дефектов в программном обеспечении, составлять описания ошибок, найденных с помощью этих же тестов.

ПРИЛОЖЕНИЯ

Приложение А – Исправный код реализации АС «Операции со строкой» на языке GoLang.

Приложение Б – Unit-тесты для определения ошибки в АС «Простой калькулятор»

Приложение А – Исправный код реализации АС «Операции со строкой» на языке GoLang.

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "time"
  "unicode"
)

type Str string

func main() {
  in := bufio.NewReader(os.Stdin)
  var str Str
  fmt.Print("Введите строку: ")
  fmt.Fscan(in, &str)
  for {
    fmt.Print("1-Разворот регистра\n2-Обратная строка \n3-Разделить строку \n4-Удалить
цифры \n5-Удалить буквы \n6-Выход \n0-Обновить строку \nВыберите дальнейшее действие:
")
    var d int
    fmt.Fscan(in, &d)

    switch d {
    case 1:
      fmt.Printf("Разворот регистра: %s\n", str.Flip())
    case 2:
      fmt.Printf("Обратная строка: %s\n", str.Reverse())
    case 3:
      var n int
      fmt.Print("Введите количество символов для разделения: ")
      fmt.Fscan(in, &n)
      fmt.Printf("Разделённая строка: \n%s\n", str.Del(n))
    case 4:
      fmt.Printf("Удаление цифр: %s\n", str.OnlySim())
    case 5:
      fmt.Printf("Удаление букв: %s\n", str.OnlyDigit())
    case 6:
      fmt.Println("Завершение...")
      time.Sleep(1 * time.Second)
      os.Exit(0)
    case 0:
      fmt.Print("Введите новую строку: ")
      fmt.Fscan(in, &str)
    default:
      fmt.Println("Нет такой команды.")
    }

    time.Sleep(1 * time.Second)
```

```
  }
}

func (s Str) OnlyDigit() string {
  var result []rune
  for _, r := range s {
    if unicode.IsDigit(r) {
      result = append(result, r)
    }
  }
  return string(result)
}

func (s Str) OnlySim() string {
  var result []rune
  for _, r := range s {
    if !unicode.IsDigit(r) {
      result = append(result, r)
    }
  }
  return string(result)
}

func (s Str) Del(n int) string {
  res := ""
  count := 0
  for _, ch := range s {
    if count == n {
      count = 0
      res += "\n"
    }
    res += string(ch)
    count++
  }
  return res
}

func (s Str) Reverse() string {
  res := ""
  for _, ch := range s {
    res = string(ch) + res
  }
  return res
}

func (s Str) Flip() string {
  runes := []rune(s)
  for i, r := range runes {
    if unicode.IsLower(r) {
      runes[i] = unicode.ToUpper(r)
```

Продолжение приложения А

```
    } else if unicode.IsUpper(r) {
      runes[i] = unicode.ToLower(r)
    }
  }
  return string(runes)
}
```

Приложение Б – Unit-тесты для определения ошибки в АС «Простой калькулятор»

```
package main

import (
  "testing"
)

func TestAdd(t *testing.T) {
  type args struct {
    a float64
    b float64
  }
  tests := []struct {
    name string
    args args
    want float64
  }{
    {
      "Test1",
      args{12, 7},
      19,
    },
    {
      "Test2",
      args{-2, 5},
      3,
    },
    {
      "Test3",
      args{99, -7},
      92,
    },
  }
  for _, tt := range tests {
    t.Run(tt.name, func(t *testing.T) {
      if got := Add(tt.args.a, tt.args.b); got != tt.want {
        t.Errorf("Add() = %v, want %v", got, tt.want)
      }
    })
  }
}

func TestSubtract(t *testing.T) {
  type args struct {
    a float64
    b float64
  }
  tests := []struct {
    name string
    args args
    want float64
```

Продолжение приложения Б

```
  }{
    {
      "Test1",
      args{12, 7},
      5,
    },
    {
      "Test2",
      args{-2, 5},
      -7,
    },
    {
      "Test3",
      args{99, -7},
      106,
    },
  }
  for _, tt := range tests {
    t.Run(tt.name, func(t *testing.T) {
      if got := Subtract(tt.args.a, tt.args.b); got != tt.want {
        t.Errorf("Subtract() = %v, want %v", got, tt.want)
      }
    })
  }
}

func TestMultiply(t *testing.T) {
  type args struct {
    a float64
    b float64
  }
  tests := []struct {
    name string
    args args
    want float64
  }{
    {
      "Test1",
      args{12, 5},
      60,
    },
    {
      "Test2",
      args{-2, 5},
      -10,
    },
    {
      "Test3",
      args{123123, 0},
      0,
```

```
    },
  }
  for _, tt := range tests {
    t.Run(tt.name, func(t *testing.T) {
      if got := Multiply(tt.args.a, tt.args.b); got != tt.want {
        t.Errorf("Multiply() = %v, want %v", got, tt.want)
      }
    })
  }
}

func TestDivide(t *testing.T) {
  type args struct {
    a float64
    b float64
  }
  tests := []struct {
    name    string
    args    args
    want    float64
    wantErr bool
  }{
    {
      "Test1",
      args{12, 3},
      4,
      false,
    },
    {
      "Test2",
      args{12, 0},
      0,
      true,
    },
    {
      "Test3",
      args{5, 2},
      2.5,
      false,
    },
  }
  for _, tt := range tests {
    t.Run(tt.name, func(t *testing.T) {
      got, err := Divide(tt.args.a, tt.args.b)
      if (err != nil) != tt.wantErr {
        t.Errorf("Divide() error = %v, wantErr %v", err, tt.wantErr)
        return
      }
      if got != tt.want {
        t.Errorf("Divide() = %v, want %v", got, tt.want)
```

Продолжение приложения Б

```
      }
    })
  }
}

func TestMod(t *testing.T) {
  type args struct {
    a float64
  }
  tests := []struct {
    name string
    args args
    want float64
  }{
    {
      "Test1",
      args{12},
      12,
    },
    {
      "Test2",
      args{-12},
      12,
    },
  }
  for _, tt := range tests {
    t.Run(tt.name, func(t *testing.T) {
      if got := Mod(tt.args.a); got != tt.want {
        t.Errorf("Mod() = %v, want %v", got, tt.want)
      }
    })
  }
}
```