МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации

**

**ЛАБОРАТОРНАЯ РАБОТА №2**

**по дисциплине: «Программирование»**

**на тему: «Классификация языков программирования»**

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБx-420», «АВТФ»,  *Никифоров Максим Владимирович*  «\_02\_» \_апреля\_ 2025г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Ассистент*  *Исаев Глеб Андреевич*  «\_\_\_» \_\_\_\_\_\_ 2025г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc194485900)

[Ход выполнения 4](#_Toc194485901)

[Задание №1 (вариант 5). 4](#_Toc194485902)

[Задание №2 (Вариант 1). 16](#_Toc194485903)

[Задание 3 (Вариант 21). 31](#_Toc194485904)

[Вывод 43](#_Toc194485905)

# ВВЕДЕНИЕ

Цели и задачи работы: изучение функций ввода-вывода данных, программирования вычисления значения выражения.

Написание программ по вариантам (5, 1, 21) на языках программирования:

1. C++
2. Kotlin
3. C#
4. PHP
5. Rust
6. Go
7. TypeScript
8. Swift
9. JavaScript
10. Python
11. Java
12. Ruby
13. Assembly (одно задание на выбор).

# Ход выполнения

## Задание №1 (вариант 5).

Студенту не хочется выполнять лабораторную работу. Он должен начать выполнять ее немедленно, чтобы уложиться в срок. Задание - напечатать строку. За раз он может выполнить одну из следующих операций:

- добавить символ в конце строки;

- добавить копию текущей строки в конец (можно применить единожды).

Помогите студенту найти минимальное количество операций,

необходимых для ввода заданной строки.

1. C++
2. #include <iostream>
3. #include <string>
4. using namespace std;
5. int minOperations(const string& s) {
6. int n = s.length();
7. int maxI = -1;
8. // Перебираем возможные длины подстроки, начиная с наибольшей
9. for (int i = n / 2 - 1; i >= 0; i--) {
10. int substrLen = i + 1;
11. if (2 \* substrLen > n) {
12. continue;
13. }
14. // Проверяем, совпадает ли первая подстрока с последующей
15. if (s.substr(0, substrLen) == s.substr(substrLen, substrLen)) {
16. maxI = i;
17. break;
18. }
19. }
20. // Возвращаем минимальное количество операций
21. return maxI != -1 ? n - maxI : n;
22. }
23. int main() {
24. //тут получаем строку от пользователя, обрабатываем, выводим
25. string s;
26. cout << "Введите строку: ";
27. getline(cin, s);
28. cout << "Результат: " << minOperations(s) << endl;
29. return 0;
30. }

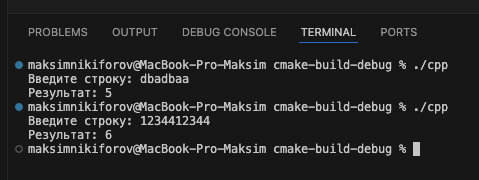
Тестирование программы (Рисунок 1)

Рисунок 1 – Тесты программы на разных входных данных

2) Kotlin

fun minOperations(s: String): Int {

val n = s.length

var maxI = -1

// Перебираем возможные длины подстроки, начиная с наибольшей

for (i in (n / 2 - 1) downTo 0) {

val substrLen = i + 1

if (2 \* substrLen > n) continue

// Проверяем, совпадает ли первая подстрока с последующей

if (s.substring(0, substrLen) == s.substring(substrLen, 2 \* substrLen)) {

maxI = i

break

}

}

// Возвращаем минимальное количество операций

return if (maxI != -1) n - maxI else n

}

fun main() {

//тут получаем строку от пользователя, обрабатываем, выводим

print("Введите строку: ")

val input = readLine()? ?: ""

println("Результат: ${minOperations(input)}")

}

Тестирование программы (Рисунок 2)

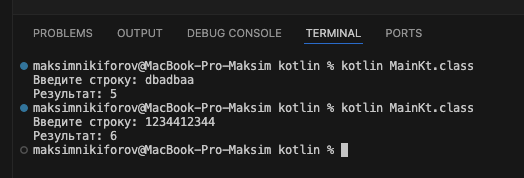


Рисунок 2 ­– Тесты программы на разных входных данных

3) C#

using System;

class Program {

static int MinOperations(string s) {

int n = s.Length;

int maxI = -1;

// Перебираем возможные длины подстроки, начиная с наибольшей

for (int i = n / 2 - 1; i >= 0; i--) {

int substrLen = i + 1;

if (2 \* substrLen > n) {

continue;

}

// Проверяем, совпадает ли первая подстрока с последующей

if (s.Substring(0, substrLen) == s.Substring(substrLen, substrLen)) {

maxI = i;

break;

}

}

// Возвращаем минимальное количество операций

return maxI != -1 ? n - maxI : n;

}

static void Main() {

//тут получаем строку от пользователя, обрабатываем, выводим

Console.Write("Введите строку: ");

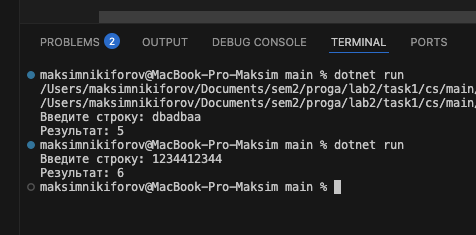
string s = Console.ReadLine();

Console.WriteLine("Результат: " + MinOperations(s));

}

}

Тестирование программы (Рисунок 3)

Рисунок 3 – Тесты программы на разных входных данных

4) PHP

<?php

function minOperations($s) {

$n = strlen($s);

$max\_i = -1;

// Перебираем возможные длины подстроки, начиная с наибольшей

for ($i = intdiv($n, 2) - 1; $i >= 0; $i--) {

$substr\_len = $i + 1;

if (2 \* $substr\_len > $n) {

continue;

}

// Проверяем, совпадает ли первая подстрока с последующей

if (substr($s, 0, $substr\_len) === substr($s, $substr\_len, $substr\_len)) {

$max\_i = $i;

break;

}

}

// Возвращаем минимальное количество операций

return $max\_i !== -1 ? $n - $max\_i : $n;

}

//тут получаем строку от пользователя, обрабатываем, выводим

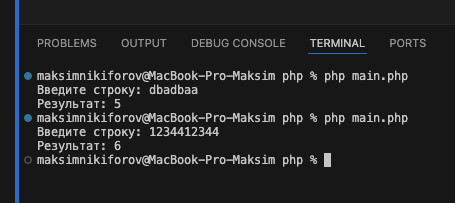
echo "Введите строку: ";

$s = trim(fgets(STDIN));

echo "Результат: " . minOperations($s) . "\n";

?>

Тестирование программы (Рисунок 4).

Рисунок 4 – Тесты программы на разных входных данных

5) Rust

use std::io;

fn min\_operations(s: &str) -> usize {

let n = s.len();

let mut max\_i = None;

// Перебираем возможные длины подстроки, начиная с наибольшей

for i in (0..n / 2).rev() {

let substr\_len = i + 1;

if 2 \* substr\_len > n {

continue;

}

// Проверяем, совпадает ли первая подстрока с последующей

if &s[0..substr\_len] == &s[substr\_len..2 \* substr\_len] {

max\_i = Some(i);

break;

}

}

// Возвращаем минимальное количество операций

match max\_i {

Some(i) => n - i,

None => n,

}

}

fn main() {

//тут получаем строку от пользователя, обрабатываем, выводим

let mut input = String::new();

println!("Введите строку: ");

io::stdin().read\_line(&mut input).expect("Ошибка чтения ввода");

let input = input.trim();

let result = min\_operations(input);

println!("Результат: {}", result);

}

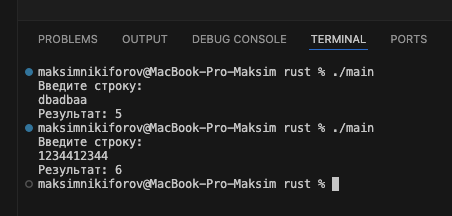
Тестирование программы (Рисунок 5).

Рисунок 5 – Тесты программы на разных входных данных

6) Go

package main

import (

"bufio"

"fmt"

"os"

"strings"

)

func minOperations(s string) int {

n := len(s)

maxI := -1

// Перебираем возможные длины подстроки, начиная с наибольшей

for i := n/2 - 1; i >= 0; i-- {

substrLen := i + 1

if 2\*substrLen > n {

continue

}

// Проверяем, совпадает ли первая подстрока с последующей

if s[:substrLen] == s[substrLen:2\*substrLen] {

maxI = i

break

}

}

// Возвращаем минимальное количество операций

if maxI != -1 {

return n - maxI

}

return n

}

func main() {

//тут получаем строку от пользователя, обрабатываем, выводим

reader := bufio.NewReader(os.Stdin)

fmt.Print("Введите строку: ")

input, \_ := reader.ReadString('\n')

input = strings.TrimSpace(input)

fmt.Printf("Результат: %d\n", minOperations(input))

}

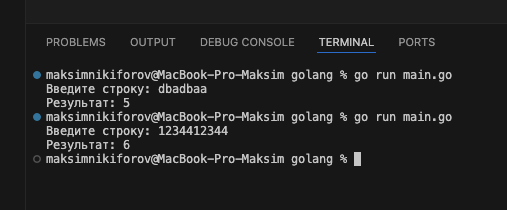
Тестирование программы (Рисунок 6).

Рисунок 6 – Тесты программы на разных входных данных

7) TypeScript

const prompter = require('prompt-sync')();

function minOperations(s) {

let n = s.length;

let max\_i = -1;

// Перебираем возможные длины подстроки, начиная с наибольшей

for (let i = Math.floor(n / 2) - 1; i >= 0; i--) {

let substr\_len = i + 1;

if (2 \* substr\_len > n) continue;

// Проверяем, совпадает ли первая подстрока с последующей

if (s.slice(0, substr\_len) === s.slice(substr\_len, 2 \* substr\_len)) {

max\_i = i;

break;

}

}

// Возвращаем минимальное количество операций

return max\_i !== -1 ? n - max\_i : n;

}

//тут получаем строку от пользователя, обрабатываем, выводим

let s = prompter("Введите строку: ");

console.log(minOperations(s));

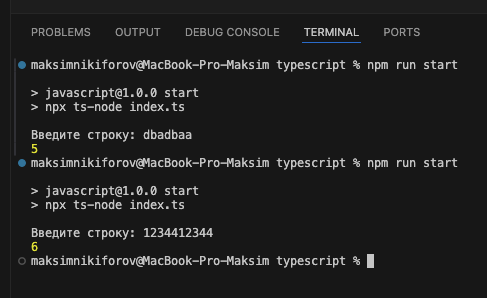
Тестирование программы (Рисунок 7).

Рисунок 7 – Тесты программы на разных входных данных

8) Swift

func minOperations(\_ s: String) -> Int {

let n = s.count

var maxI = -1

let chars = Array(s)

// Перебираем возможные длины подстроки, начиная с наибольшей

for i in stride(from: n / 2 - 1, through: 0, by: -1) {

let substrLen = i + 1

if 2 \* substrLen > n { continue }

// Проверяем, совпадает ли первая подстрока с последующей

if chars[0..<substrLen] == chars[substrLen..<(2 \* substrLen)] {

maxI = i

break

}

}

// Возвращаем минимальное количество операций

return maxI != -1 ? n - maxI : n

}

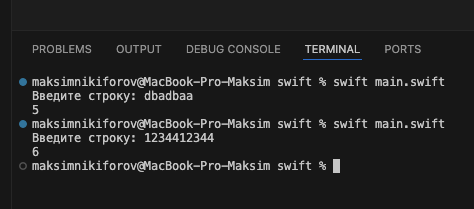
//тут получаем строку от пользователя, обрабатываем, выводим

print("Введите строку: ", terminator: "")

let s = readLine() ?? ""

print(minOperations(s))

Тестирование программы (Рисунок 8).

Рисунок 8 – Тесты программы на разных входных данных

9) JavaScript

const prompt = require('prompt-sync')();

function minOperations(s) {

let n = s.length;

let max\_i = -1;

// Перебираем возможные длины подстроки, начиная с наибольшей

for (let i = Math.floor(n / 2) - 1; i >= 0; i--) {

let substr\_len = i + 1;

if (2 \* substr\_len > n) continue;

// Проверяем, совпадает ли первая подстрока с последующей

if (s.slice(0, substr\_len) === s.slice(substr\_len, 2 \* substr\_len)) {

max\_i = i;

break;

}

}

// Возвращаем минимальное количество операций

return max\_i !== -1 ? n - max\_i : n;

}

//тут получаем строку от пользователя, обрабатываем, выводим

let s = prompt("Введите строку: ");

console.log(minOperations(s));

Тестирование программы (Рисунок 9).

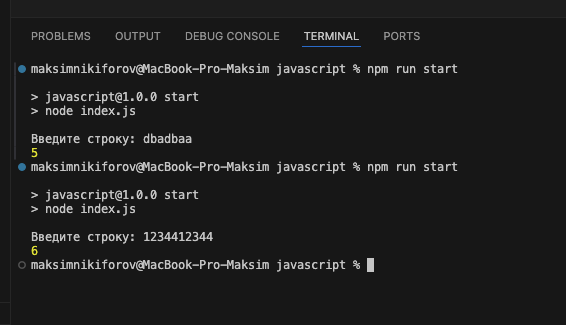


Рисунок 9 – Тесты программы на разных входных данных

10) Python

def min\_operations(s):

n = len(s)

max\_i = -1

# Проверяем возможные позиции i, начиная с наибольшей

for i in range((n // 2 - 1), -1, -1):

substr\_len = i + 1

if 2 \* substr\_len > n:

continue

if s[:substr\_len] == s[substr\_len : 2 \* substr\_len]:

max\_i = i

break # Нашли максимальный подходящий i

return n - max\_i if max\_i != -1 else n

#тут получаем строку от пользователя, обрабатываем, выводим

s = input("Введите строку: ")

print(min\_operations(s))

Тестирование программы (Рисунок 10).

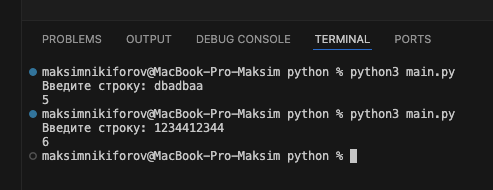


Рисунок 10 – Тесты программы на разных входных данных

11) Java

import java.util.Scanner;

public class main {

public static int minOperations(String s) {

int n = s.length();

int maxI = -1;

// Перебираем возможные длины подстроки, начиная с наибольшей

for (int i = n / 2 - 1; i >= 0; i--) {

int substrLen = i + 1;

if (2 \* substrLen > n) {

continue;

}

// Проверяем, совпадает ли первая подстрока с последующей

if (s.substring(0, substrLen).equals(s.substring(substrLen, 2 \* substrLen))) {

maxI = i;

break;

}

}

// Возвращаем минимальное количество операций

return maxI != -1 ? n - maxI : n;

}

public static void main(String[] args) {

//тут получаем строку от пользователя, обрабатываем, выводим

Scanner scanner = new Scanner(System.in);

System.out.print("Введите строку: ");

String s = scanner.nextLine().trim();

System.out.println("Результат: " + minOperations(s));

scanner.close();

}

}

Тестирование программы (Рисунок 11).

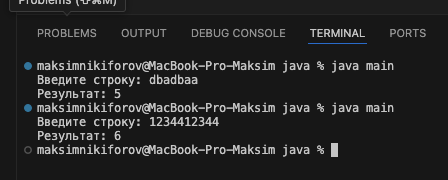


Рисунок 11 – Тесты программы на разных входных данных

12) Ruby

def min\_operations(s)

n = s.length

max\_i = -1

# Перебираем возможные длины подстроки, начиная с наибольшей

(n / 2 - 1).downto(0) do |i|

substr\_len = i + 1

next if 2 \* substr\_len > n

# Проверяем, совпадает ли первая подстрока с последующей

if s[0...substr\_len] == s[substr\_len...(2 \* substr\_len)]

max\_i = i

break

end

end

# Возвращаем минимальное количество операций

max\_i != -1 ? n - max\_i : n

end

#тут получаем строку от пользователя, обрабатываем, выводим

puts "Введите строку:"

s = gets.chomp

puts "Результат: #{min\_operations(s)}"

Тестирование программы (Рисунок 12).

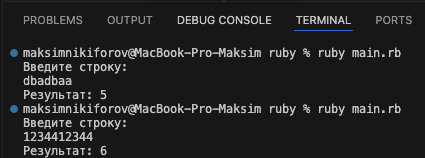


Рисунок 12 – Тесты программы на разных входных данных

## Задание №2 (Вариант 1).

Адрес электронной почты (email) — индивидуальный адрес, который присваивается каждому пользователю для получения электронных писем. Email состоит из уникального имени пользователя почты, значка «@», обозначающего, что это email-адрес, и доменного имени.

Ограничения:

— Имя пользователя может состоять из 6–30 знаков и содержать буквы, цифры и символы;

— Имя пользователя может содержать буквы латинского алфавита (a– z), цифры (0–9) и точки (.);

— Запрещено использовать амперсанд (&), знаки равенства (=) и сложения (+), скобки (<>), запятую (,), символ подчеркивания (\_), апостроф ('), дефис (-) и несколько точек подряд;

— Имя пользователя может начинаться и заканчиваться любым разрешенным символом, кроме точки (.).

Необходимо вычислить количество уникальных адресов при условии, что точка, оставленная внутри имени пользователя, будет проигнорирована, а использование \* в имени пользователя игнорирует набор символов после (в пределах имени пользователя).

1. C++
2. #include <iostream>
3. #include <string>
4. #include <set>
5. using namespace std;
6. // Проверяем, соответствует ли email требованиям
7. bool checkLegality(const string& address) {
8. int atPos = address.find('@');
9. if (atPos == string::npos) return false; // Если нет '@' — не email
10. string user = address.substr(0, atPos); // Часть до '@'
11. string domN = address.substr(atPos, address.length());
12. if(domN.find('.') == string::npos) return false;
13. // Длина имени пользователя должна быть 6-30 символов
14. if (user.length() < 6 || user.length() > 30) return false;
15. // Запрещенные символы в имени
16. string bannedChars = "&=+<>,\_'-";
17. for (char c : user) {
18. if (bannedChars.find(c) != string::npos) return false;
19. }
20. // Проверка на точки в начале/конце и двойные точки
21. if (user.front() == '.' || user.back() == '.') return false;
22. if (user.find("..") != string::npos) return false;
23. return true; // если все ок
24. }
25. // Убираем звездочки и точки из имени
26. string sanitizeName(string address) {
27. size\_t atPos = address.find('@');
28. string user = address.substr(0, atPos);
29. string domain = address.substr(atPos + 1);
30. // Обрезаем все после звездочки
31. size\_t starPos = user.find('\*');
32. if (starPos != string::npos) {
33. user = user.substr(0, starPos);
34. }
35. // Удаляем все точки
36. user.erase(remove(user.begin(), user.end(), '.'), user.end());
37. return user + "@" + domain; // Собираем email обратно
38. }
39. int main() {
40. int n;
41. cout << "Введите количество адресов: ";
42. cin >> n;
43. cin.ignore(); // Чистим буфер
44. set<string> adrs; // Храним уникальные адреса
45. for (int i = 0; i < n; i++) {
46. cout << "Введите адрес: ";
47. string adr;
48. getline(cin, adr);
49. if (checkLegality(adr)) { // Если email валидный
50. adr = sanitizeName(adr); // Чистим его
51. adrs.insert(adr); // Добавляем в set (автоматическая уникальность)
52. }
53. }
54. cout << adrs.size() << endl; // Выводим количество уникальных адресов
55. }

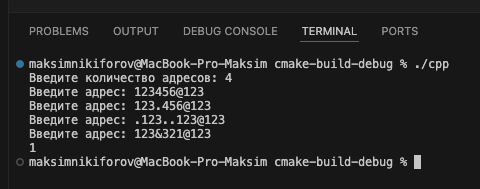
Тестирование программы (Рисунок 13).

Рисунок 13 – Тестирование программы

2) Kotlin

import java.util.HashSet

import java.util.Scanner

fun checkLegality(address: String): Boolean {

// Адрес должен содержать символ '@'

if (!address.contains("@")) {return false}

val user = address.split("@", limit = 2)[0]

if(!address.split("@", limit = 2)[1].contains('.')) return false

// Длина имени пользователя от 6 до 30 символов

if (user.length < 6 || user.length > 30) {

return false

}

// Запрещённые символы

for (i in 0 until user.length) {

val c = user[i]

if ("&=+<>,\_'-".indexOf(c) != -1) {

return false

}

}

// Имя пользователя не должно начинаться или заканчиваться точкой,

// а также содержать две точки подряд

if (user.startsWith(".") || user.endsWith(".")) { return false }

if (user.contains("..")) { return false }

return true

}

// Функция очистки адреса: обрезается всё после символа '\*'

// и удаляются все точки в имени пользователя

fun sanitizeName(address: String): String {

val parts = address.split("@", limit = 2)

val user = parts[0].split("\*", limit = 2)[0].replace(".", "")

return user + "@" + parts[1]

}

fun main() {

val scanner = Scanner(System.`in`)

print("Введите количество адресов: ")

val n = scanner.nextLine().trim().toInt()

val adrs: HashSet<String> = HashSet()

for (i in 0 until n) {

print("Введите адрес: ")

val rawAdr = scanner.nextLine().trim()

// Сначала проверяем исходный адрес

if (checkLegality(rawAdr)) {

// Если адрес корректен, выполняем очистку

val adr = sanitizeName(rawAdr)

adrs.add(adr)

}

}

println(adrs.size)

}

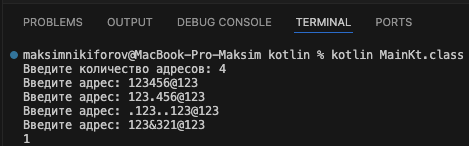
Тестирование программы (Рисунок 14).

Рисунок 14 – Тестирование программы

3) C#

using System;

using System.Collections.Generic;

public class Program {

public static bool checkLegality(string address) {

// Адрес должен содержать символ '@'

if (!address.Contains("@")) return false;

string[] parts = address.Split(new char[] { '@' }, 2);

string user = parts[0];

if(!parts[1].Contains(".")) return false;

// Длина имени пользователя от 6 до 30 символов

if (user.Length < 6 || user.Length > 30) return false;

// Запрещённые символы

for (int i = 0; i < user.Length; i++) {

char c = user[i];

if ("&=+<>,\_'-".IndexOf(c) != -1) return false;

}

// Имя пользователя не должно начинаться или заканчиваться точкой,

// а также содержать две точки подряд

if (user.StartsWith(".") || user.EndsWith(".")) return false;

if (user.Contains("..")) return false;

return true;

}

// Функция очистки адреса: обрезается всё после символа '\*'

// и удаляются все точки в имени пользователя

public static string sanitizeName(string address) {

string[] parts = address.Split(new char[] { '@' }, 2);

string user = parts[0].Split(new char[] { '\*' }, 2)[0].Replace(".", "");

return user + "@" + parts[1];

}

public static void Main(string[] args) {

Console.Write("Введите количество адресов: ");

int n = int.Parse(Console.ReadLine());

List<string> adrs = new List<string>();

for (int i = 0; i < n; i++) {

Console.Write("Введите адрес: ");

string rawAdr = Console.ReadLine();

// Сначала проверяем исходный адрес

if (checkLegality(rawAdr)) {

// Если адрес корректен, выполняем очистку

string adr = sanitizeName(rawAdr);

adrs.Add(adr);

}

}

Console.WriteLine(adrs.Count);

}

}

Тестирование программы (Рисунок 15).

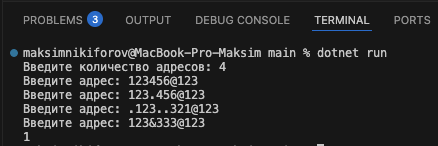


Рисунок 15 – Тестирование программы

4) PHP

<?php

function checkLegality($address) {

// Проверяем, что адрес содержит '@'

if (!str\_contains($address, '@')) {

return false;

}

// Извлекаем имя пользователя

list($user, $domain) = explode('@', $address, 2);

if(!str\_contains($domain,'.')) {return false;}

// Длина имени пользователя должна быть от 6 до 30 символов

if (strlen($user) < 6 || strlen($user) > 30) {

return false;

}

// Проверяем наличие недопустимых символов в имени пользователя

$invalidChars = "&=+<>,\_'-";

for ($i = 0; $i < strlen($user); $i++) {

if (str\_contains($invalidChars, $user[$i])) {

return false;

}

}

// Имя пользователя не должно начинаться или заканчиваться точкой, а также содержать две точки подряд

if (str\_starts\_with($user, '.') || str\_ends\_with($user, '.')) { return false; }

if (str\_contains($user, '..')) { return false; }

return true;

}

function sanitizeName($address) {

// Разбиваем адрес на имя пользователя и домен

list($user, $domain) = explode('@', $address, 2);

// Обрезаем часть после символа '\*', затем удаляем все точки

$user = explode('\*', $user, 2)[0];

$user = str\_replace('.', '', $user);

return $user . '@' . $domain;

}

$a = (int) readline("Введите количество адресов: ");

$adrs = array();

for ($n = 0; $n < $a; $n++) {

$rawAdr = trim(readline("Введите адрес: "));

// Сначала проверяем исходный адрес на корректность

if (checkLegality($rawAdr)) {

// Если адрес корректен, выполняем санитайзинг

$adr = sanitizeName($rawAdr);

// Добавляем адрес, если он ещё не содержится в массиве

if (!in\_array($adr, $adrs)) {

$adrs[] = $adr;

}

}

}

print(count($adrs) . "\n");?>

Тестирование программы (Рисунок 16).

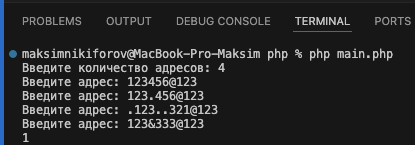


Рисунок 16 – Тестирование программы

5) Rust

use std::collections::HashSet;

use std::io;

fn check\_legality(address: &str) -> bool {

let user = address.split('@').next().unwrap\_or("");

if(!address.split('@').next().contains('.')) {return false}

if user.len() < 6 || user.len() > 30 {

return false;

}

let forbidden\_chars = "&=+<>,\_'-";

if user.chars().any(|c| forbidden\_chars.contains(c)) {

return false;

}

if user.starts\_with('.') || user.ends\_with('.') || user.contains("..") {

return false;

}

true

}

fn sanitize\_name(address: &str) -> String {

if let Some((user, domain)) = address.split\_once('@') {

let user = user.split('\*').next().unwrap\_or("").replace('.', "");

return format!("{}@{}", user, domain);

}

address.to\_string()

}

fn main() {

let mut input = String::new();

println!("Введите количество адресов: ");

io::stdin().read\_line(&mut input).expect("Ошибка ввода");

let n: usize = input.trim().parse().expect("Введите корректное число");

let mut adrs: HashSet<String> = HashSet::new();

for \_ in 0..n {

let mut raw\_adr = String::new();

println!("Введите адрес: ");

io::stdin().read\_line(&mut raw\_adr).expect("Ошибка ввода");

let raw\_adr = raw\_adr.trim().to\_lowercase();

if check\_legality(&raw\_adr) {

let adr = sanitize\_name(&raw\_adr);

adrs.insert(adr);

}

}

println!("{}", adrs.len());

}

Тестирование программы (Рисунок 17).

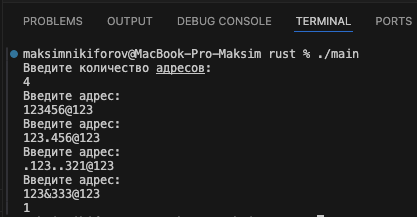


Рисунок 17 – Тестирование программы

6) Go

package main

import (

"bufio"

"fmt"

"os"

"strings"

)

func checkLegality(address string) bool {

// Разделяем адрес на части

parts := strings.Split(address, "@")

if len(parts) != 2 {return false}

user := parts[0]

if !strings.Contains(parts[1],".") {return false}

// Проверка длины имени пользователя

if len(user) < 6 || len(user) > 30 {return false}

// Проверка наличия запрещённых символов

for \_, char := range user {

if strings.ContainsRune("&=+<>,\_'-", char) {return false}

}

// Проверка точек в имени пользователя

if strings.HasPrefix(user, ".") || strings.HasSuffix(user, ".") {return false}

if strings.Contains(user, "..") {return false}

return true

}

// Очистка имени пользователя

func sanitizeName(address string) string {

parts := strings.Split(address, "@")

user := strings.Split(parts[0], "\*")[0] // Отрезаем часть после '\*'

user = strings.ReplaceAll(user, ".", "") // Удаляем все точки

return user + "@" + parts[1]

}

func main() {

reader := bufio.NewReader(os.Stdin)

var n int

fmt.Print("Введите количество адресов: ")

fmt.Scan(&n)

adrs := make(map[string]bool)

for i := 0; i < n; i++ {

fmt.Print("Введите адрес: ")

adr, \_ := reader.ReadString('\n')

adr = strings.TrimSpace(strings.ToLower(adr)) // Убираем пробелы и переводим в нижний регистр

// Сначала проверяем на корректность, потом чистим

if checkLegality(adr) {

adr = sanitizeName(adr)

adrs[adr] = true

}

}

fmt.Println(len(adrs))

}

Тестирование программы (Рисунок 18)

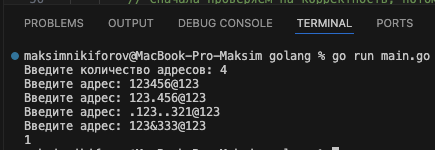


Рисунок 18 – Тестирование программы

7) TypeScript

const prompter = require('prompt-sync')();

const checkLegality = (address: string): boolean => {

if (!address.includes('@')) return false;

const user = address.split('@')[0];

// Длина юзнейма части должна быть от 6 до 30 символов

if (user.length < 6 || user.length > 30) return false;

// Проверяем, что в юзнейм нет запрещённых символов

for (let i = 0; i < user.length; i++) {

if ("&=+<>,\_'-".includes(user[i])) return false;

}

if (user.startsWith('.') || user.endsWith('.')) return false; //юз нейм не начинается и не заканчивается точкой,

if (user.includes('..')) return false; // и не содержит двух точек подряд

return true;

};

// отбрасываем всё, что идёт после символа '\*', и удаляем точки из локальной части

const sanitizeName = (address: string): string => {

const [user, domain] = address.split('@');

// Отбрасываем часть после символа '\*'

const sanitizedUser = user.split('\*', 1)[0].replace(/\./g, '');

return `${sanitizedUser}@${domain}`;

};

const adrs = new Set<string>();

const n = +prompter('Введите количество адресов: ');

for (let i = 0; i < n; i++) {

const input = prompter('Введите адрес: ');

// Сначала проверяем, соответствует ли адрес правилам

if (checkLegality(input)) {

// Если адрес корректен, то санитайзим его и добавляем в множество

const adr = sanitizeName(input);

adrs.add(adr);

}

}

console.log(adrs.size);

Тестирование программы (Рисунок 19)

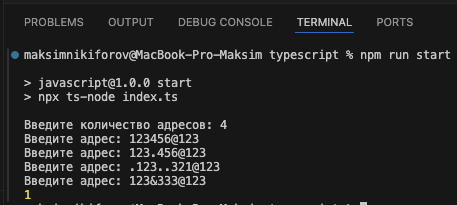


Рисунок 19 – Тестирование программы

8) Swift

import Foundation

func checkLegality(address: String) -> Bool {

if !address.contains("@") { return false }// Проверяем, что адрес содержит символ '@'

// вычленяем юзер

let parts = address.split(separator: "@", maxSplits: 1, omittingEmptySubsequences: false)

let user = String(parts[0])

if !String(parts[1]).contains(".") {return false}

// Длина от 6 до 30 символов

if user.count < 6 || user.count > 30 { return false }

// Проверяем, что в локальной части нет запрещённых символов

for ch in user {

if "&=+<>,\_'-".contains(ch) {return false}

}

if user.hasPrefix(".") || user.hasSuffix(".") { return false }

if user.contains("..") { return false }

return true

}

// Отбрасываем всё, что идёт после символа '\*', и удаляем точки из локальной части.

func sanitizeName(address: String) -> String {

let parts = address.split(separator: "@", maxSplits: 1, omittingEmptySubsequences: false)

let userPart = String(parts[0])

let domainPart = String(parts[1])

// Отбрасываем часть локальной части после символа '\*'

let userPt1 = userPart.split(separator: "\*", maxSplits: 1, omittingEmptySubsequences: false).first ?? ""

let finalUser = userPt1.replacingOccurrences(of: ".", with: "") // Убираем все точки

return "\(finalUser)@\(domainPart)"

}

print("Введите количество адресов: ", terminator: "")

var adrs = Set<String>()

let n = Int(readLine() ?? "") ?? 0

// Считываем адреса, проверяем их легальность, затем санитайзим и сохраняем уникальные адреса

for \_ in 0..<n {

print("Введите адрес: ", terminator: "")

if let input = readLine() {

if checkLegality(address: input) {

let sanitized = sanitizeName(address: input)

adrs.insert(sanitized)

}

}

}

print(adrs.count)// Выводим количество уникальных корректных адресов

Тестирование программы (Рисунок 20)

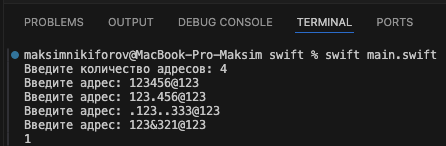


Рисунок 20 – Тестирование программы

9) JavaScript

const prompter = require('prompt-sync')();

const checkLegality = (address: string): boolean => {

if (!address.includes('@')) return false;

const[user,domN] = address.split('@');

if(!domN.includes(".")) return false;

// Длина юзнейма части должна быть от 6 до 30 символов

if (user.length < 6 || user.length > 30) return false;

// Проверяем, что в юзнейм нет запрещённых символов

for (let i = 0; i < user.length; i++) {

if ("&=+<>,\_'-".includes(user[i])) return false;

}

if (user.startsWith('.') || user.endsWith('.')) return false; //юз нейм не начинается и не заканчивается точкой,

if (user.includes('..')) return false; // и не содержит двух точек подряд

return true;

};

// отбрасываем всё, что идёт после символа '\*', и удаляем точки из локальной части

const sanitizeName = (address: string): string => {

const [user, domain] = address.split('@');

// Отбрасываем часть после символа '\*'

const sanitizedUser = user.split('\*', 1)[0].replace(/\./g, '');

return `${sanitizedUser}@${domain}`;

};

const adrs = new Set<string>();

const n = +prompter('Введите количество адресов: ');

for (let i = 0; i < n; i++) {

const input = prompter('Введите адрес: ');

// Сначала проверяем, соответствует ли адрес правилам

if (checkLegality(input)) {

// Если адрес корректен, то санитайзим его и добавляем в множество

const adr = sanitizeName(input);

adrs.add(adr);

}

}

console.log(adrs.size);

Тестирование программы (Рисунок 21)

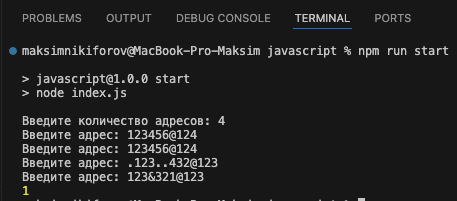


Рисунок 21 – Тестирование программы

10) Python

def check\_legality(address):

# Разбиваем email на имя пользователя и проверяем длину

user, domN = address.split('@')

if not '.' in domN: return False

if not (6 <= len(user) <= 30): return False

# Проверяем, нет ли запрещенных символов

if any(symb in "&=+<>,\_'-" for symb in user): return False

#Проверяем точки в начале/конце и двойные точки

if user.startswith('.') or user.endswith('.') or ".." in user:

return False

return True # Если все ок

def sanitize\_name(address):

# Разбиваем email на имя и домен

user, domain = address.split('@', 1)

# Убираем все после звездочки и точки в имени

user = user.split('\*', 1)[0].replace('.', '')

return f"{user}@{domain}" # Собираем обратно

adrs = set()

# Запрашиваем количество адресов

for \_ in range(int(input("Введите количество адресов: "))):

# Читаем и чистим ввод

raw\_adr = input("Введите адрес: ").strip().lower()

if check\_legality(raw\_adr): # Если email валидный

adr = sanitize\_name(raw\_adr) # Чистим его

adrs.add(adr) # Добавляем в set

print(len(adrs))

Тестирование программы (Рисунок 22).

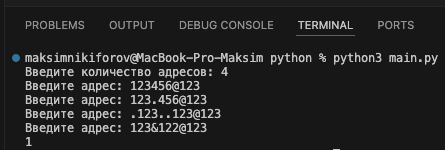


Рисунок 22 – Тестирование программы

11) Java

import java.util.HashSet;

import java.util.Scanner;

class Main {

public static boolean checkLegality(String address) {

// Адрес должен содержать символ '@'

if (!address.contains("@")) {return false;}

String user = address.split("@", 2)[0];

String domN = address.split("@", 2)[1];

if(!domN.contains(".")) {return false;}

// Длина имени пользователя от 6 до 30 символов

if (user.length() < 6 || user.length() > 30) {return false;}

// Запрещённые символы

for (int i = 0; i < user.length(); i++) {

char c = user.charAt(i);

if ("&=+<>,\_'-".indexOf(c) != -1) {return false;}

}

// Имя пользователя не должно начинаться или заканчиваться точкой,

// а также содержать две точки подряд

if (user.startsWith(".") || user.endsWith(".")) {return false;}

if (user.contains("..")) {return false;}

return true;

}

// Функция очистки адреса: обрезается всё после символа '\*'

// и удаляются все точки в имени пользователя

public static String sanitizeName(String address) {

String[] parts = address.split("@", 2);

String user = parts[0].split("\\\*", 2)[0].replace(".", "");

return user + "@" + parts[1];

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Введите количество адресов: ");

int n = Integer.parseInt(scanner.nextLine().trim());

HashSet<String> adrs = new HashSet<>();

for (int i = 0; i < n; i++) {

System.out.print("Введите адрес: ");

String rawAdr = scanner.nextLine().trim().toLowerCase();

// Сначала проверяем исходный адрес

if (checkLegality(rawAdr)) {

// Если адрес корректен, выполняем очистку

String adr = sanitizeName(rawAdr);

adrs.add(adr);

}

}

System.out.println(adrs.size());

}

}

Тестирование программы (Рисунок 23).

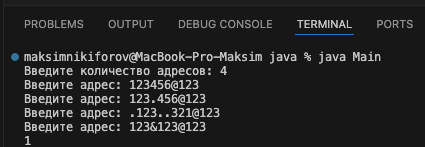


Рисунок 23 – Тестирование программы

12) Ruby

require 'set' # модуль для множеств

# Проверяем валидность email адреса

def check\_legality(address)

user = address.split('@').first# Выделяем часть до @ (имя пользователя)

domN = address.split('@').second

return false if not domN.include?(".")

return false if user.length < 6 || user.length > 30# Проверяем длину имени (6-30 символов)

return false if user.chars.any? { |c| "&=+<>,\_'-".include?(c) }# Проверяем на наличие запрещенных символов

# Проверяем точки в начале/конце и двойные точки

return false if user.start\_with?('.') || user.end\_with?('.')

return false if user.include?('..')

true # Если все проверки пройдены

end

# Нормализуем email (удаляем лишнее)

def sanitize\_name(address)

user, domain = address.split('@', 2)# Разбиваем на имя и домен

user = user.split('\*').first.gsub('.', '')# Удаляем все после звездочки и все точки

"#{user}@#{domain}"# Собираем emil обратно

end

# Основная программа

print "Введите количество адресов: "

n = gets.to\_i # Читаем количество адресов

adrs = Set.new # Создаем множество для хранения уникальных адресов

n.times do

# Читаем и чистим ввод

raw\_adr = gets.chomp.strip

# Если адрес валидный

if check\_legality(raw\_adr)

# Нормализуем его

adr = sanitize\_name(raw\_adr)

# Добавляем в множество (автоматическая уникальность)

adrs.add(adr)

end

end

puts adrs.size

Тестирование программы (Рисунок 24).

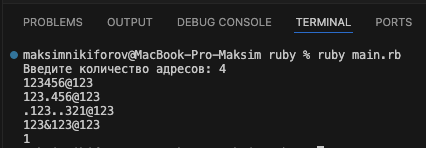


Рисунок 24 – Тестирование программы

## Задание 3 (Вариант 21).

Вывести номера чисел из последовательности, у которых сумма цифр меньше их произведения (например, у числа 87 сумма цифр равна 15, а произведение 56).

1) C++

#include <iostream>

using namespace std;

int main() {

int n; // Количество чисел для обработки

cout << "Введите количество чисел: ";

cin >> n;

for (int i = 0; i < n; i++) {

int num; // Текущее число

cout << "Введите число: ";

cin >> num;

int digSum = 0, digWork = 1; // Сумма и произведение цифр

// Разбиваем число на цифры

while (num > 0) {

int dig = num % 10; // Получаем последнюю цифру

num /= 10; // Убираем последнюю цифру из числа

digSum += dig; // Добавляем цифру к сумме

digWork \*= dig; // Умножаем на произведение

}

// Проверяем условие и выводим индекс, если оно выполняется

if (digSum < digWork) {

cout << i << endl; // Выводим индекс числа (не само число!)

}

}

return 0;

}

Тестирование программы (Рисунок 25).

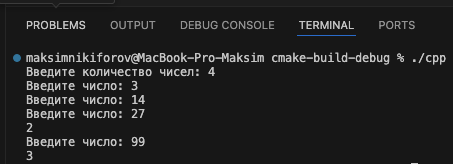


Рисунок 25 – Тестирование программы

2) Kotlin

import java.util.Scanner // Для ввода данных

fun main() {

val myObj = Scanner(System.`in`) // Создаем сканер для чтения ввода

print("Введите количество чисел: ")

val n = myObj.nextLine().toInt() // Читаем количество чисел

for (i in 0 until n) {

print("Введите число: ")

var num = myObj.nextLine().toInt() // Читаем очередное число

var digSum = 0 // Будем хранить сумму цифр

var digWork = 1 // Будем хранить произведение цифр

while (num > 0) {

val dig = num % 10 // Получаем последнюю цифру

num /= 10 // Убираем обработанную цифру

digSum += dig // Добавляем к сумме

digWork \*= dig // Умножаем произведение

}

if (digSum < digWork) { // Если сумма меньше произведения

println(i) // Выводим индекс числа (не само число!)

}

}

}

Тестирование программы (Рисунок 26)

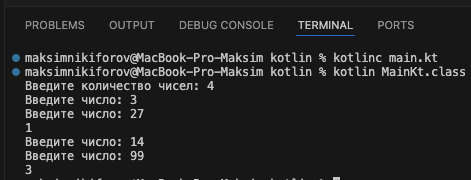


Рисунок 26 – Тестирование программы

3) C#

using System;

class Program {

static void Main(string[] args) {

Console.Write("Введите количество чисел: ");

int n = int.Parse(Console.ReadLine()); // Чтение количества чисел

for (int i = 0; i < n; i++) {

Console.Write("Введите число: ");

int num = int.Parse(Console.ReadLine()); // Чтение очередного числа

int digSum = 0; // Сумма цифр числа

int digWork = 1; // Произведение цифр числа

while (num > 0) {

int dig = num % 10; // Получение последней цифры

num = num / 10; // Удаление обработанной цифры

digSum += dig; // Накопление суммы

digWork \*= dig; // Накопление произведения

}

if (digSum < digWork) { // Проверка условия

Console.WriteLine(i); // Вывод индекса числа

}

}

}

}

Тестирование программы (Рисунок 27)

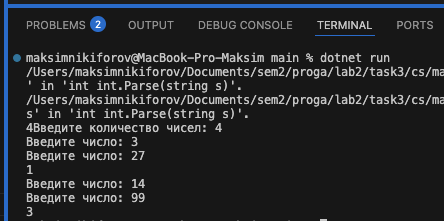


Рисунок 27 – Тестирование программы

4) PHP

<?php

$a = (int)readline("Введите количество чисел: "); // Получаем количество чисел для обработки

for($n = 0; $n < $a; $n++) {

$num = (int)readline("Введите число: "); // Читаем очередное число

$digSum = 0; // Инициализация суммы цифр (исправлено с 1 на 0)

$digWork = 1; // Инициализация произведения цифр

while ($num > 0) {

$dig = $num % 10; // Получаем последнюю цифру

$num = (int)($num / 10); // Удаляем последнюю цифру

$digSum += $dig; // Добавляем цифру к сумме

$digWork \*= $dig; // Умножаем на произведение

}

if ($digSum < $digWork) { // Проверяем условие

echo $n . "\n"; // Выводим индекс числа (не само число!)

}

}

?>

Тестирование программы (Рисунок 28)

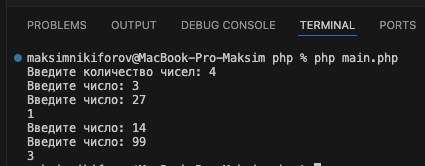


Рисунок 28 – Тестирование программы

5) Rust

fn main() {

use std::io::{stdin, stdout, Write}; // Импорт необходимых модулей ввода-вывода

print!("Введите количество чисел: ");

let \_ = stdout().flush(); // Принудительный вывод буфера (чтобы print! работал сразу)

let mut input\_line = String::new(); // Буфер для ввода

stdin().read\_line(&mut input\_line).expect("Failed to read line"); // Чтение строки

let x: u16 = input\_line.trim().parse().expect("Input not an integer"); // Парсинг числа

for n in 0..x { // Цикл по всем числам

print!("Введите число: ");

let \_ = stdout().flush(); // Снова сброс буфера

let mut number = String::new(); // Буфер для числа

stdin().read\_line(&mut number).expect("Failed to read line"); // Чтение числа

let mut num: u16 = number.trim().parse().expect("Input not an integer"); // Парсинг числа

let mut digSum = 0; // Сумма цифр

let mut digWork = 1; // Произведение цифр

while num > 0 { // Разбираем число на цифры

let dig = num % 10; // Последняя цифра

num = num / 10; // Убираем последнюю цифру

digSum += dig; // Накопление суммы

digWork \*= dig; // Накопление произведения

}

if digSum < digWork { // Проверка условия

print!("{}\n", n); // Вывод индекса числа

let \_ = stdout().flush(); // Сброс буфера вывода

}

}

}

Тестирование программы (Рисунок 29)

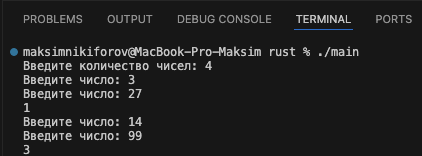


Рисунок 29 – Тестирование программы

6) Go

package main

import "fmt"

func main() {

fmt.Print("Введите количество чисел: ")

var n int

fmt.Scan(&n) // Чтение количества чисел

for i := 0; i < n; i++ { // Цикл по всем числам

fmt.Print("Введите число: ")

var num int

fmt.Scan(&num) // Чтение очередного числа

digSum := 0 // Инициализация суммы цифр

digWork := 1 // Инициализация произведения цифр

for num > 0 { // Разбор числа на цифры

dig := num % 10 // Получение последней цифры

num = num / 10 // Удаление обработанной цифры

digSum += dig // Накопление суммы

digWork \*= dig // Накопление произведения

}

if digSum < digWork { // Проверка условия

fmt.Println(i) // Вывод индекса числа (не самого числа)

}

}

}

Тестирование программы (Рисунок 30)

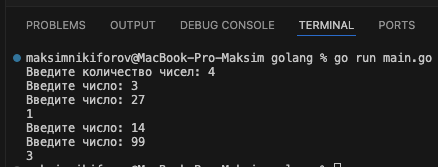


Рисунок 30 – Тестирование программы

7) TypeScript

const prompter = require('prompt-sync')(); // Инициализация вводного интерфейса

const n = +prompter('Введите количество чисел: ') // Получаем количество чисел

for(let i = 0; i < n; i++) { // Цикл обработки n чисел

let num = +prompter('Введите число: ') // Читаем очередное число

let digSum = 0; // Исправлено: сумма цифр (было 1, стало 0)

let digWork = 1; // Произведение цифр

while(num > 0) { // Разбираем число на цифры

const dig = num % 10 // Получаем последнюю цифру

num = Math.floor(num / 10) // Удаляем последнюю цифру

digSum += dig // Суммируем цифры

digWork \*= dig // Перемножаем цифры

}

digSum < digWork && console.log(i) // Если сумма < произведения - выводим индекс

}

Тестирование программы (Рисунок 31)

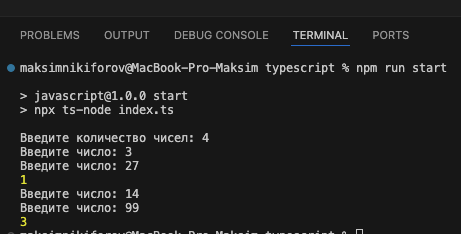


Рисунок 31 – Тестирование программы

8) Swift

print("Введите количество чисел: ", terminator: "")

let n: Int = Int(readLine() ?? "") ?? 0 // Чтение количества чисел с обработкой nil

for i in 0..<n { // Цикл по всем числам (исправлен диапазон)

print("Введите число: ", terminator: "")

var number: Int = Int(readLine() ?? "") ?? 0 // Чтение числа с обработкой nil

var digSum = 0 // Сумма цифр (исправлено с 1 на 0)

var digWork = 1 // Произведение цифр

while number > 0 { // Разбор числа на цифры

let dig = number % 10 // Получение последней цифры

number /= 10 // Удаление обработанной цифры

digSum += dig // Накопление суммы

digWork \*= dig // Накопление произведения

}

if digSum < digWork { // Проверка условия

print(i) // Вывод индекса числа

}

}

Тестирование программы (Рисунок 32)

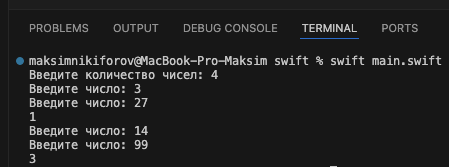


Рисунок 32 – Тестирование программы

9) JavaScript

const prompter = require('prompt-sync')(); // Инициализация вводного интерфейса

const n = +prompter('Введите количество чисел: ') // Получаем количество чисел

for(let i = 0; i < n; i++) { // Цикл обработки n чисел

let num = +prompter('Введите число: ') // Читаем очередное число

let digSum = 0; // Исправлено: сумма цифр (было 1, стало 0)

let digWork = 1; // Произведение цифр

while(num > 0) { // Разбираем число на цифры

const dig = num % 10 // Получаем последнюю цифру

num = Math.floor(num / 10) // Удаляем последнюю цифру

digSum += dig // Суммируем цифры

digWork \*= dig // Перемножаем цифры

}

digSum < digWork && console.log(i) // Если сумма < произведения - выводим индекс

}

Тестирование программы (Рисунок 33)

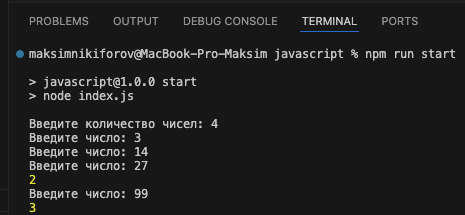


Рисунок 33 – Тестирование программы

10) Python

import math # Импорт математического модуля (хотя в данном коде не используется)

n = int(input("Введите количество чисел: ")) # Получаем количество чисел для обработки

for i in range(n): # Цикл по всем числам (упрощен диапазон)

num = int(input("Введите число: ")) # Читаем число

digSum = 0 # Инициализация суммы цифр

digWork = 1 # Инициализация произведения цифр

while num > 0: # Разбираем число на цифры

dig = num % 10 # Получаем последнюю цифру

num = num // 10 # Удаляем последнюю цифру

digSum += dig # Суммируем цифры

digWork \*= dig # Перемножаем цифры

if digSum < digWork: # Проверяем условие

print(i) # Выводим индекс числа (не само число!)

Тестирование программы (Рисунок 34)

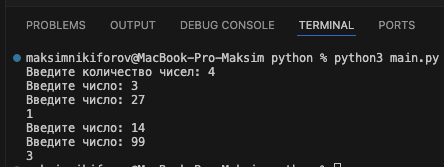


Рисунок 34 – Тестирование программы

11) Java

import java.util.Scanner; // Импорт класса для ввода данных

class Main {

public static void main(String[] args) {

Scanner myObj = new Scanner(System.in); // Создание объекта Scanner для чтения ввода

System.out.print("Введите количество чисел: ");

int n = Integer.parseInt(myObj.nextLine());

for(int i = 0; i < n; i++) { // Цикл обработки n чисел

System.out.print("Введите число: ");

int num = Integer.parseInt(myObj.nextLine()); // Чтение числ

int digSum = 0; // Сумма цифр числа

int digWork = 1; // Произведение цифр числа

while (num > 0) { // Разбор числа на цифры

int dig = num % 10; // Получение последней цифры

num /= 10; // Удаление обработанной цифры

digSum += dig; // Накопление суммы

digWork \*= dig; // Накопление произведения

}

if(digSum < digWork) { // Проверка условия

System.out.println(i); // Вывод индекса числа (не самого числа)

}

}

}

}

Тестирование программы (Рисунок 35)

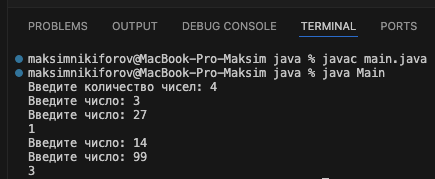


Рисунок 35 – Тестирование программы

12) Ruby

print "Введите количество чисел: "

input = gets.to\_i # Чтение количества чисел

(0...input).each do |n| # Цикл по диапазону

print "Введите число: "

num = gets.to\_i # Чтение числа

dig\_sum = 0 # Сумма цифр

dig\_work = 1 # Произведение цифр

while num > 0 # Разбор числа на цифры

dig = num % 10 # Получение последней цифры

num /= 10 # Удаление обработанной цифры

dig\_sum += dig # Накопление суммы

dig\_work \*= dig # Накопление произведения

end

puts n if dig\_sum < dig\_work # Условный вывод в одну строку

end

Тестирование программы (Рисунок 36)

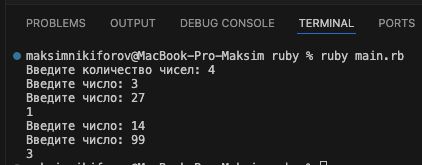


Рисунок 36 – Тестирование программы

13) Assembly

.global \_main

.align 2

\_main:

// Вывод приглашения

mov x0, #1 // stdout

adr x1, prompt // адрес строки

mov x2, #prompt\_len // длина строки

mov x16, #4 // syscall write

svc #0

// Чтение числа

mov x0, #0 // stdin

sub sp, sp, #16 // выделяем 16 байт на стеке

mov x1, sp // буфер для ввода

mov x2, #16 // размер буфера

mov x16, #3 // syscall read

svc #0

// Преобразование ASCII в число и вычисление суммы и произведения цифр

mov x19, #0 // sum = 0

mov x20, #1 // product = 1

mov x21, sp // указатель на буфер

convert\_loop:

ldrb w22, [x21], #1 // загружаем символ

cmp w22, #10 // проверка на перенос строки (Enter)

beq convert\_done

sub w22, w22, #'0' // ASCII в число

add x19, x19, x22 // sum += digit

mul x20, x20, x22 // product \*= digit

b convert\_loop

convert\_done:

// Сравнение суммы и произведения

cmp x19, x20

bge print\_zero

print\_one:

mov x0, #1 // stdout

adr x1, one\_msg // строка "1"

mov x2, #2 // длина строки

mov x16, #4 // syscall write

svc #0

b exit

print\_zero:

mov x0, #1 // stdout

adr x1, zero\_msg // строка "0"

mov x2, #2 // длина строки

mov x16, #4 // syscall write

svc #0

exit:

// Завершение программы

mov x0, #0

mov x16, #1

svc #0

prompt:

.ascii "Введите число: "

prompt\_len = . - prompt

one\_msg:

.ascii "1\n"

zero\_msg:

.ascii "0\n"

Тестирование работы программы (Рисунок 37)

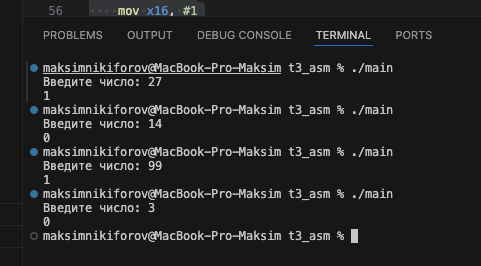


Рисунок 37 – Тестирование программы

# Вывод

В результате выполнения данной лабораторной работы я достиг цели по изучению новых языков программирования, работы с вводом-выводом, создания простых программ для вычислений значений выражений на 12 языках программирования.

**Ссылка на github**: https://github.com/Kefircheggg/nstu\_lab2