МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра защиты информации

**

**ЛАБОРАТОРНАЯ РАБОТА №3**

**по дисциплине: «Программирование»**

**на тему: «Программирование циклического вычислительного процесса»**

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБ-420», «АВТФ»,  *Никифоров Максим Владимирович*  «14» \_\_мая\_ 2025г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Ассистент кафедры ЗИ*  *Исаев Глеб Андреевич*  «\_\_\_» \_\_\_\_\_\_ 2025г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2025

Оглавление

[Введение 3](#_Toc198080717)

[Ход выполнения 4](#_Toc198080718)

[Задание 1. 4](#_Toc198080719)

[Реализация на C++. 4](#_Toc198080720)

[Реализация на Go. 5](#_Toc198080721)

[Задание 2. 7](#_Toc198080722)

[Реализация на C++. 7](#_Toc198080723)

[Реализация на Go. 14](#_Toc198080724)

[Задание 3. 21](#_Toc198080725)

[Реализация на C++. 21](#_Toc198080726)

[Реализация на Go. 22](#_Toc198080727)

[Задание 4 24](#_Toc198080728)

[Реализация на C++. 24](#_Toc198080729)

[Реализация на Go. 25](#_Toc198080730)

# Введение

Цели и задачи работы: изучение циклических алгоритмов, операторов цикла, программирование циклического вычислительного процесса.

Задание к работе: Реализовать циклический вычислительный процесс. Самостоятельно решить задачи в соответствии с индивидуальным вариантом на двух языках программирования (С++ и другой (по согласованию с преподавателем)).

Задание 1. Вычислить и вывести на экран или в файл в виде таблицы значения функции, заданной графически, на интервале от Xнач до Xкон с шагом dx. Интервал и шаг задать таким образом, чтобы проверить все ветви программы.

Задание 2. Реализовать тесты на простоту согласно заданию.

Задание 3. Математическая сумма.

Задание 4. Увлекательная игра.

Методика выполнения работы:

1. определить типы используемых в программе данных;

2. описать переменные;

3. написать функции ввода-вывода;

4. разработать алгоритм решения задачи по индивидуальному заданию;

5. написать и отладить программу с вводом-выводом информации;

6. протестировать работу программы на различных исходных данных;

7. изменить формат вывода, проверить работу программы при другом формате вывода.

# Ход выполнения

## Задание 1.

### Реализация на C++.

#include <iostream>

#include <iomanip>

using namespace std;

double func(double x) {

if (x <= -7) return 0;

if (x>-7 && x < -3) return x+7;

if (x>=-3 && x<-2) return 4;

if (x>=-2 && x < 2) return x\*x;

if (x >= 2 && x < 4) return -2\*x+8;

if (x>=4) return 0;

}

int main() {

double x0, x1, dx;

cout << "Введите x0: ";

cin >> x0;

cout << "Введите x1: ";

cin >> x1;

cout << "Введите dx: ";

cin >> dx;

cout << " x y\n";

for (double i = x0; i <= x1; i+=dx) {

cout << setw(3) << i << setw(3) <<func(i) << endl;

}

return 0;

}

Тестирование работы программы (Рисунок 1).

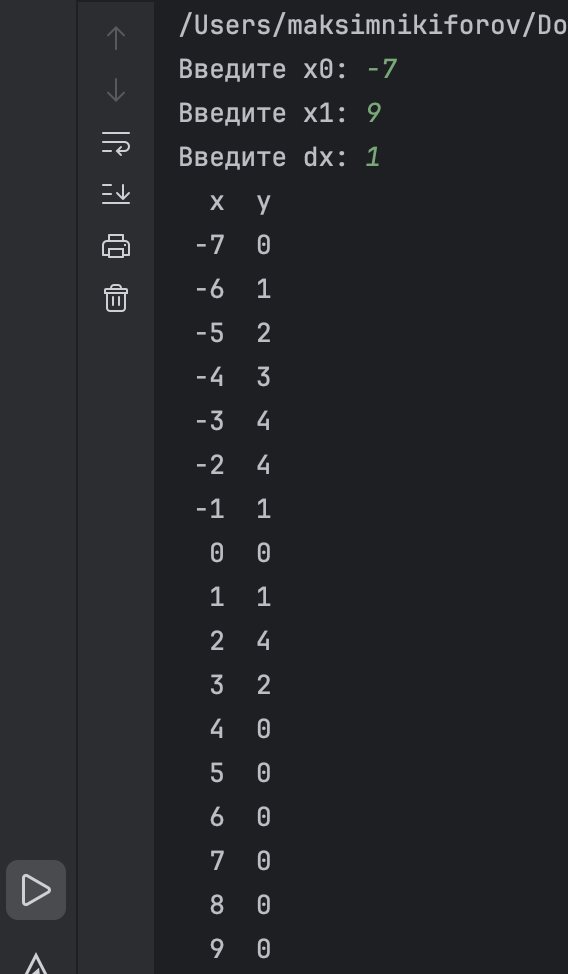


Рисунок 1 – Тестирование работы программы

### Реализация на Go.

package main

import (

"fmt"

)

func f(x float64) float64 {

if x <= -7 {

return 0

}

if x > -7 && x < -3 {

return x + 7

}

if x >= -3 && x < -2 {

return 4

}

if x >= -2 && x < 2 {

return x \* x

}

if x >= 2 && x < 4 {

return -2\*x + 8

}

if x >= 4 {

return 0

}

return 0

}

func main() {

var x0, x1, dx float64

fmt.Print("Введите x0: ")

fmt.Scan(&x0)

fmt.Print("Введите x1: ")

fmt.Scan(&x1)

fmt.Print("Введите dx: ")

fmt.Scan(&dx)

fmt.Println(" x y")

for x := x0; x <= x1; x += dx {

fmt.Printf("%3.1f %3.1f\n", x, f(x))

}

}

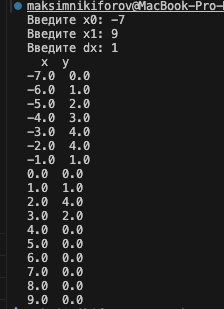
Тестирование работы программы (Рисунок 2).

Рисунок 2 – Тестирование программы на Go.

## Задание 2.

### Реализация на C++.

Программа–меню, все три теста на выбор.

#include <algorithm>

#include <cmath>

#include <iostream>

#include <random>

#include <tuple>

#include <vector>

#include <iomanip>

using namespace std;

// генератор случайных чисел

mt19937 mt\_rand(random\_device{}());

// генерация рандомных чисел с равной вероятностью

int uniRand(int a, int b) { return uniform\_int\_distribution<int>(a, b)(mt\_rand); }

double uniRandD(int a, int b) { return uniform\_real\_distribution<double>(a, b)(mt\_rand); }

// возведение в степень по модулю через двоичное разложение

long powModDecopmose(long a, long x, long p) {

long res = 1; // результат единичка чтобы было на что умножать

a%=p; // приводим по модулю

while (x > 0) { // пока от числа вообще что-то осталось

// если бит (с младшего) - 1 домножаем результат на остаток от деления a^2^i на кольце

if (x % 2 == 1) { res = (res \* a) % p; }

a = (a \* a) % p; // остаток считаем каждый раз чтобы не считать страшные степени

x /= 2; // ну и делим само число

}

return res;

}

// решето эратосфена

vector<int> eratSieve(const int limit) {

vector<bool> is\_prime(limit, true);

is\_prime[0] = is\_prime[1] = false;

for (int i = 2; i \* i < limit; ++i) {

if (is\_prime[i]) {

for (int j = i \* i; j < limit; j += i) {

is\_prime[j] = false;

}

}

}

vector<int> primes;

for (int i = 2; i < limit; ++i) {

if (is\_prime[i]) {

primes.push\_back(i);

}

}

return primes;

}

// Построение числа n

pair<int, vector<int>> builderTest(const vector<int> &primes, int bit\_len) {

int max\_index = 0; // индекс используемый для выбора простых чисел из primes

int max\_pow = 1; // степень до которой мы будем возводить числа из primes

while (max\_index < primes.size() && primes[max\_index] < pow(2, bit\_len - 1))

max\_index++;

while (pow(2, max\_pow) < pow(2, bit\_len - 1))

max\_pow++;

int m = 1;

vector<int> q;

while (true) {

int num = uniRand(0, max\_index - 1);

int power = uniRand(1, max\_pow);

if (pow(primes[num], power) != 0) { // проверка и добавление числа

if ((double)m \* pow(primes[num], power) < INT32\_MAX) {

m \*= pow(primes[num], power);

q.push\_back(primes[num]);

}

}

if (m > pow(2, bit\_len - 2)) { // проверка размера числа

if (m >= pow(2, bit\_len - 1)) {

m = 1;

q.clear();

} else {

break;

}

}

}

int n = 2 \* m + 1;

return make\_pair(n, q);

}

// Тест Миллера

int millerTest(int n, int t, const vector<int> &q) {

vector<int> a; // числа для проверки 1<aj<n

while (a.size() != t) {

int aj = uniRand(2, n - 1);

if (find(a.begin(), a.end(), aj) == a.end()) {

a.push\_back(aj);

}

}

for (int aj : a) { // проверяем степень числа

if (powModDecopmose(aj, n - 1, n) != 1) {

return 0;

}

}

for (int qi : q) {

bool millerCrit = true;

for (int aj : a) {

if (powModDecopmose(aj, (n - 1) / qi, n) != 1) {

millerCrit = false;

break;

}

}

if (millerCrit) {

return 0;

}

}

return 1;

}

int poklingtonTest(int n, int t, const vector<int> &q) {

vector<int> a;

int aj;

while (a.size() != t) {

aj = uniRand(1, n);

if (find(a.begin(), a.end(), aj) == a.end()) {

a.push\_back(aj);

}

}

// 1ая проверка

for (int aj : a) {

if (powModDecopmose(aj, n - 1, n) != 1) {

return 0;

}

}

// 2ая проверка

for (int aj : a) {

bool flag = true;

for (int qi : q) {

if (qi != 0 && powModDecopmose(aj, (n - 1) / qi, n) == 1) {

flag = false;

break;

}

}

if (flag) {

return 1;

}

}

return 0;

}

// Алгоритм перехода от меньшего простого числа к большему

int build\_new\_from\_old(vector<int> prime, int bit) {

int q;

int max\_index = 0;

int p;

for (; prime[max\_index] < pow(2, bit / 2); max\_index++)

;

while (true) {

q = prime[uniRand(0, max\_index)];

if (q > 0 && q <= pow(2, bit / 2) - 1)

break;

}

while (true) {

double epsilon = uniRandD(0, 1);

double n0 = (pow(2, bit - 1) / q) + (pow(2, bit - 1) \* epsilon / q);

int n = ceil(n0);

if ((int)n % 2 == 1)

n++;

for (int u = 0; true; u += 2) {

p = (n + u) \* q + 1;

if (p > pow(2, bit))

break;

if (powModDecopmose(2, p - 1,p) == 1 && powModDecopmose(2, n + u, p) != 1)

return p;

}

}

}

void resultsPrint(const vector<int>& nums, const vector<bool>& nums\_good, const vector<int>& declined) {

cout << "Простое число\tРезультат теста\tОтвергнутые\n";

for (size\_t i = 0; i < nums.size(); ++i) {

string testRes = nums\_good[i] ? "+" : "-";

cout << setw(13) << nums[i] << setw(18) << testRes << setw(12) << declined[i]

<< endl;

}

}

int main() {

int maxPrime = 500;

vector<int> primes = eratSieve(maxPrime);

int len\_bit;

cout << "Введите длину желаемого числа (1-32 бит): ";

cin >> len\_bit;

int testType;

cout << "Укажите желаемый тип теста (1 – Миллера, 2 – Поклингтона, 3 – ГОСТ): ";

cin >> testType;

switch (testType) {

case 1: {

cout << "Получение простых чисел заданной длины на основе теста Миллера\n";

vector<int> q;

int n;

int k = 0;

int prob;

vector<int> nums;

vector<bool> nums\_good;

vector<int> declined;

while (nums.size() < 10) {

tie(n, q) = builderTest(primes, len\_bit);

prob = millerTest(n, 10, q);

if (prob == 1) {

if (find(nums.begin(), nums.end(), n) == nums.end()) {

nums.push\_back(n);

prob = millerTest(n, 1, q);

nums\_good.push\_back(prob);

declined.push\_back(k);

k = 0;

}

} else {

k++;

}

}

resultsPrint(nums,nums\_good,declined);

break;

}

case 2: {

cout << "Получение простых чисел заданной длины на основе теста Поклингтона\n";

vector<int> q;

int n;

int k = 0;

int probability;

vector<int> nums;

vector<bool> nums\_good;

vector<int> declined;

while (nums.size() != 10) {

tie(n, q) = builderTest(primes, len\_bit);

probability = poklingtonTest(n, 10, q);

if (probability == 1) {

if (find(nums.begin(), nums.end(), n) == nums.end()) {

nums.push\_back(n);

nums\_good.push\_back(poklingtonTest(n, 1, q));

declined.push\_back(k);

k = 0;

}

}

else {

k++;

}

}

resultsPrint(nums, nums\_good, declined);

break;

}

case 3: {

cout << "Генерация простых чисел заданной длины на основе ГОСТ 34.10-94\n";

vector<int> res;

while (res.size() != 10) {

int p = build\_new\_from\_old(primes, len\_bit);

if (find(res.begin(), res.end(), p) == res.end()) {

res.push\_back(p);

}

}

for (int i = 1; i <= res.size(); i++) {

cout << setw(3) << i << setw(8) << res[i-1] << endl;

}

break;

}

default: {

cout << "Так не пойдет, завершаю исполнение\n";

return 0;

}

}

}

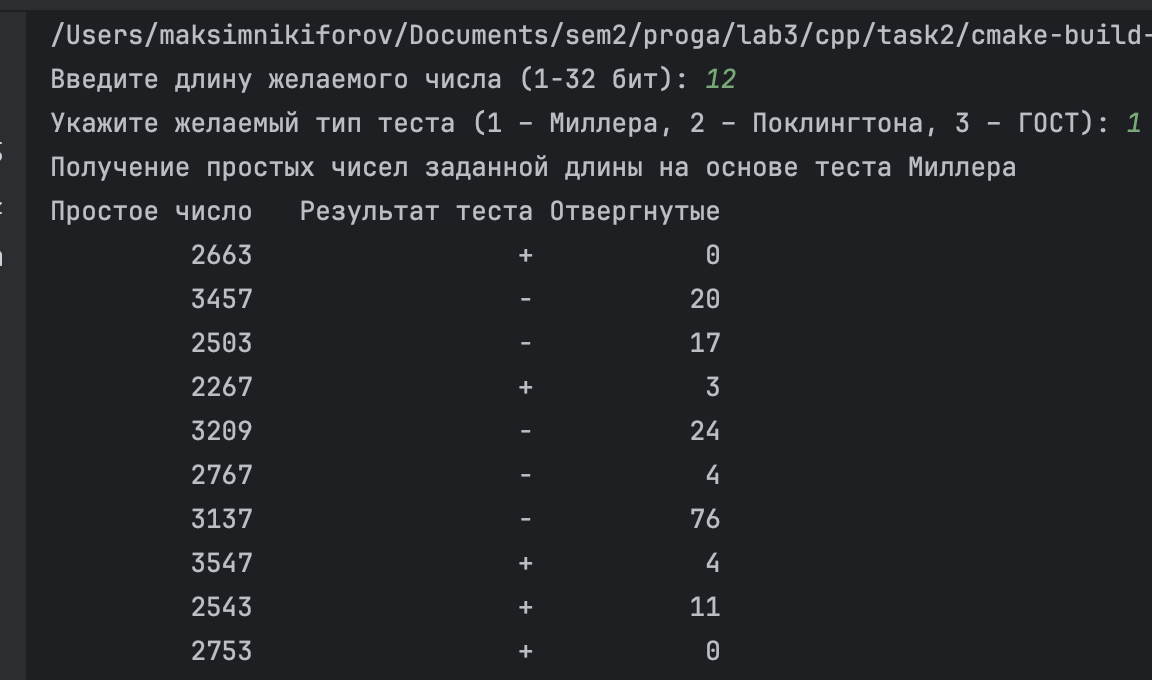
Тестирование работы программы (Рисунок 3, Рисунок 4, Рисунок 5).

Рисунок 3 – тестирование работы программы (тест Миллера)

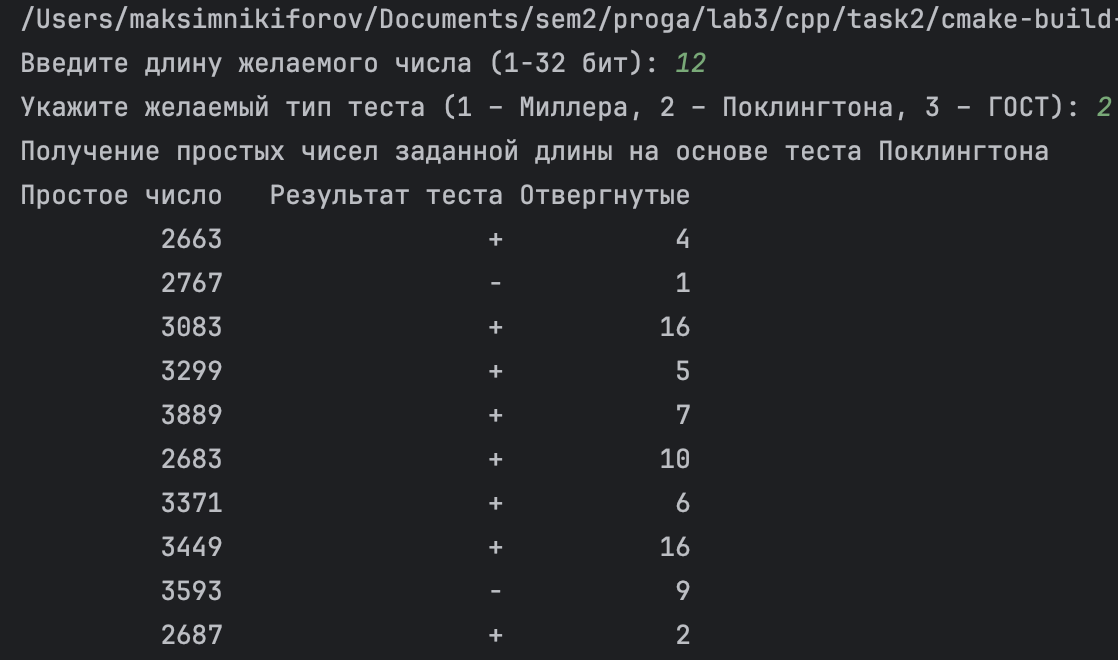


Рисунок 4 – тестирование работы программы (тест Поклингтона)

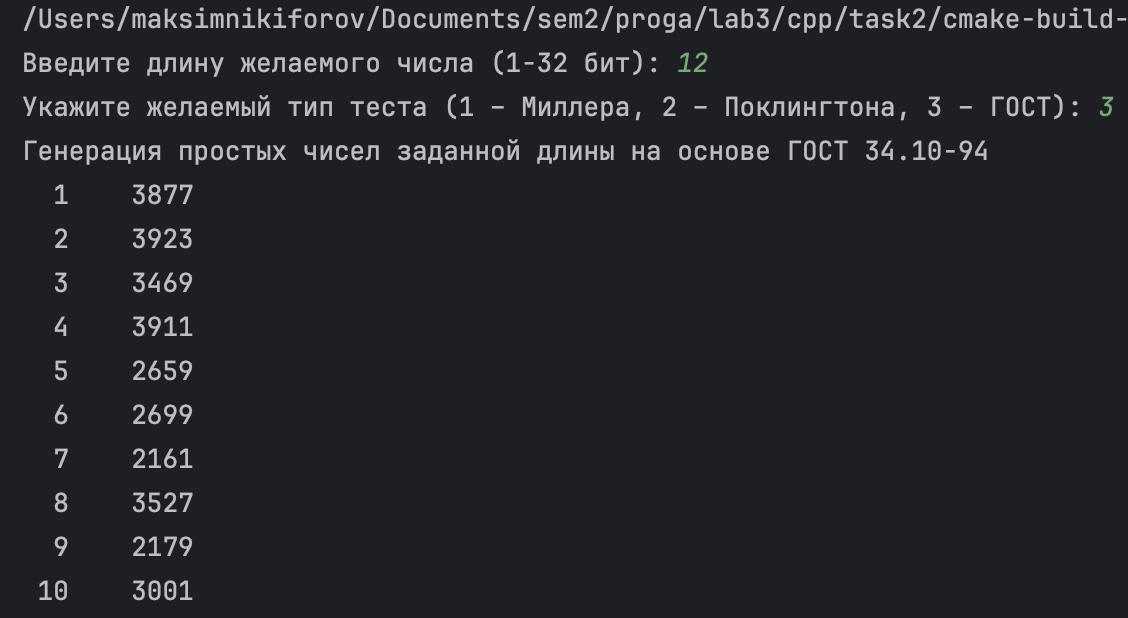


Рисунок 5 – тестирование работы программы (генерация по ГОСТ)

### Реализация на Go.

package main

import (

"fmt"

"math"

"math/rand"

"sort"

"time"

)

var rnd = rand.New(rand.NewSource(time.Now().UnixNano()))

func uniRand(a, b int) int {

return rnd.Intn(b-a+1) + a

}

func uniRandD(a, b int) float64 {

return rnd.Float64()\*float64(b-a) + float64(a)

}

// возводим в степень по модулю

func powModDec(a, x, p int) int {

res := 1

a %= p

for x > 0 {

if x%2 == 1 {

res = (res \* a) % p

}

a = (a \* a) % p

x /= 2

}

return res

}

// делаем решето эратосфена

func eratSieve(limit int) []int {

isPrime := make([]bool, limit)

for i := range isPrime {

isPrime[i] = true

}

isPrime[0], isPrime[1] = false, false

for i := 2; i\*i < limit; i++ {

if isPrime[i] {

for j := i \* i; j < limit; j += i {

isPrime[j] = false

}

}

}

var primes []int

for i := 2; i < limit; i++ {

if isPrime[i] {

primes = append(primes, i)

}

}

return primes

}

func builderTest(primes []int, bitLen int) (int, []int) {

maxIndex := 0

for maxIndex < len(primes) && primes[maxIndex] < int(math.Pow(2, float64(bitLen-1))) {

maxIndex++

}

maxPow := 1

for math.Pow(2, float64(maxPow)) < math.Pow(2, float64(bitLen-1)) {

maxPow++

}

m := 1

var q []int

for {

num := uniRand(0, maxIndex-1)

power := uniRand(1, maxPow)

powVal := int(math.Pow(float64(primes[num]), float64(power)))

if powVal != 0 && float64(m)\*float64(powVal) < float64(math.MaxInt32) {

m \*= powVal

q = append(q, primes[num])

}

if float64(m) > math.Pow(2, float64(bitLen-2)) {

if float64(m) >= math.Pow(2, float64(bitLen-1)) {

m = 1

q = nil

} else {

break

}

}

}

n := 2\*m + 1

return n, q

}

func millerTest(n, t int, q []int) bool {

var a []int

for len(a) != t {

aj := uniRand(2, n-1)

if !contains(a, aj) {

a = append(a, aj)

}

}

for \_, aj := range a {

if powModDec(aj, n-1, n) != 1 {

return false

}

}

for \_, qi := range q {

pass := false

for \_, aj := range a {

if powModDec(aj, (n-1)/qi, n) != 1 {

pass = true

break

}

}

if !pass {

return false

}

}

return true

}

func poklingtonTest(n, t int, q []int) bool {

var a []int

for len(a) != t {

aj := uniRand(1, n)

if !contains(a, aj) {

a = append(a, aj)

}

}

for \_, aj := range a {

if powModDec(aj, n-1, n) != 1 {

return false

}

}

for \_, aj := range a {

flag := true

for \_, qi := range q {

if qi != 0 && powModDec(aj, (n-1)/qi, n) == 1 {

flag = false

break

}

}

if flag {

return true

}

}

return false

}

func buildNewFromOld(primes []int, bit int) int {

maxIndex := 0

for primes[maxIndex] < int(math.Pow(2, float64(bit/2))) {

maxIndex++

}

var q int

for {

q = primes[uniRand(0, maxIndex)]

if q > 0 && q <= int(math.Pow(2, float64(bit/2))-1) {

break

}

}

for {

eps := uniRandD(0, 1)

n0 := math.Pow(2, float64(bit-1))/float64(q) + (math.Pow(2, float64(bit-1))\*eps)/float64(q)

n := int(math.Ceil(n0))

if n%2 == 1 {

n++

}

for u := 0; ; u += 2 {

p := (n+u)\*q + 1

if float64(p) > math.Pow(2, float64(bit)) {

break

}

if powModDec(2, p-1, p) == 1 && powModDec(2, n+u, p) != 1 {

return p

}

}

}

}

func resultsPrint(nums []int, good []bool, declined []int) {

fmt.Println("Простое число\tРезультат теста\tОтвергнутые")

for i := range nums {

res := "-"

if good[i] {

res = "+"

}

fmt.Printf("%13d%18s%12d\n", nums[i], res, declined[i])

}

}

func contains(slice []int, x int) bool {

for \_, val := range slice {

if val == x {

return true

}

}

return false

}

func main() {

var lenBit int

fmt.Print("Введите длину желаемого числа (1-32 бит): ")

fmt.Scan(&lenBit)

fmt.Print("Укажите желаемый тип теста (1 – Миллера, 2 – Поклингтона, 3 – ГОСТ): ")

var testType int

fmt.Scan(&testType)

primes := eratSieve(500)

switch testType {

case 1:

fmt.Println("Получение простых чисел заданной длины на основе теста Миллера")

var nums []int

var good []bool

var declined []int

k := 0

for len(nums) < 10 {

n, q := builderTest(primes, lenBit)

if millerTest(n, 10, q) {

if !contains(nums, n) {

nums = append(nums, n)

good = append(good, millerTest(n, 1, q))

declined = append(declined, k)

k = 0

}

} else {

k++

}

}

resultsPrint(nums, good, declined)

case 2:

fmt.Println("Получение простых чисел заданной длины на основе теста Поклингтона")

var nums []int

var good []bool

var declined []int

k := 0

for len(nums) < 10 {

n, q := builderTest(primes, lenBit)

if poklingtonTest(n, 10, q) {

if !contains(nums, n) {

nums = append(nums, n)

good = append(good, poklingtonTest(n, 1, q))

declined = append(declined, k)

k = 0

}

} else {

k++

}

}

resultsPrint(nums, good, declined)

case 3:

fmt.Println("Генерация простых чисел заданной длины на основе ГОСТ 34.10-94")

var res []int

for len(res) < 10 {

p := buildNewFromOld(primes, lenBit)

if !contains(res, p) {

res = append(res, p)

}

}

sort.Ints(res)

for i, v := range res {

fmt.Printf("%3d%8d\n", i+1, v)

}

default:

fmt.Println("Так не пойдет, завершаю исполнение")

}

}

Тестирование работы программы (Рисунок 6, Рисунок 7, Рисунок 8).

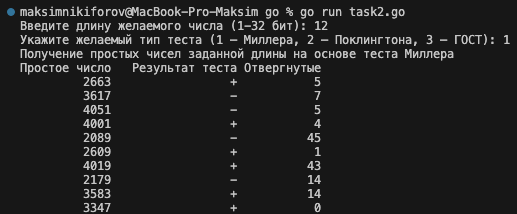


Рисунок 6 – тестирование работы программы (тест Миллера)

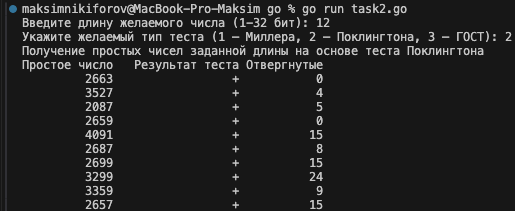


Рисунок 7 – тестирование работы программы (тест Поклингтона)

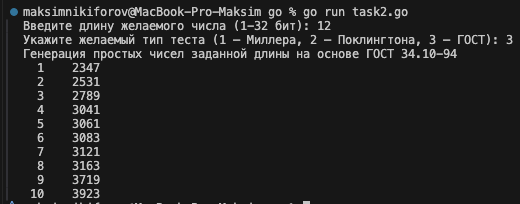


Рисунок 8 – тестирование работы программы (генерация по ГОСТ)

## Задание 3.

### Реализация на C++.

#include <cmath>

#include <iostream>

using namespace std;

int gcd(int a, int b) {

pair<int,int> X = {a,0};

pair<int,int> Y = {b,1};

while (Y.first!=0) {

int q = X.first / Y.first;

pair<int,int> T = {X.first-q\*Y.first, X.second-q\*Y.second};

X = Y; Y = T;

}

return X.first;

}

double findSeriesSum(int a, int b) {

double cur;

double sum\_past = 0, sum = 0;

cur = 1. / b;

sum\_past = cur;

sum = cur;

for (int n = 2; n < 1000; n++) {

cur = pow(n, a) / pow(b, n);

sum += cur;

if (fabs(sum - sum\_past) < 0.000001) {

return sum;

}

sum\_past = sum;

}

return -1;

}

bool is\_possibly\_rational(double x) {

for (int b = 1; b <= 10000; b++) {

int a = (int)round(x \* b);

if (fabs(x - (double)a / b) < 0.000001) {

int common\_divisor = gcd(a, b);

cout << "Сумма — рациональное число: "

<< a / common\_divisor << "/" << b / common\_divisor << endl;

return true;

}

}

return false;

}

int main() {

int a, b;

cout << "Введите целые a и b: ";

cin >> a >> b;

double serSum = findSeriesSum(a, b);

if (serSum == -1) {

cout << "Ряд расходится (бесконечная сумма)" << endl;

} else if (!is\_possibly\_rational(serSum)) {

cout << "Сумма — иррациональное число: " << serSum << endl;

}

return 0;

}

Тестирование работы программы (Рисунок 9, Рисунок 10)

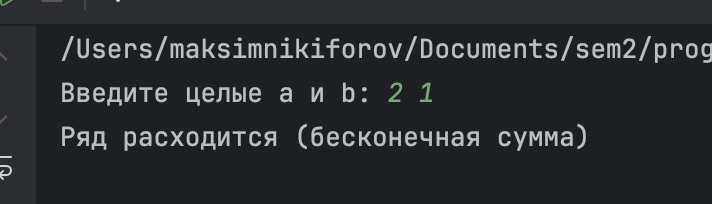


Рисунок 9 – Тестирование программы на расходящемся ряде

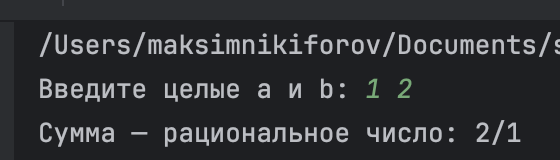


Рисунок 10 – Тестирование на сходящемся к рациональному числу ряде

### Реализация на Go.

package main

import (

"fmt"

"math"

)

// НОД по расширенному алгоритму Евклида

func gcd(a, b int) int {

X := [2]int{a, 0}

Y := [2]int{b, 1}

for Y[0] != 0 {

q := X[0] / Y[0]

T := [2]int{X[0] - q\*Y[0], X[1] - q\*Y[1]}

X = Y

Y = T

}

return X[0]

}

// считаем сумму ряда

func findSeriesSum(a, b int) float64 {

cur := 1.0 / float64(b)

sumPast := cur

sum := cur

for n := 2; n < 1000; n++ {

cur = math.Pow(float64(n), float64(a)) / math.Pow(float64(b), float64(n))

sum += cur

if math.Abs(sum-sumPast) < 0.000001 {

return sum

}

sumPast = sum

}

return -1

}

// пытаемся представить как рациональное

func isPossiblyRational(x float64) bool {

for b := 1; b <= 10000; b++ {

a := int(math.Round(x \* float64(b)))

if math.Abs(x-float64(a)/float64(b)) < 0.000001 {

commonDiv := gcd(a, b)

fmt.Printf("Сумма — рациональное число: %d/%d\n", a/commonDiv, b/commonDiv)

return true

}

}

return false

}

func main() {

var a, b int

fmt.Print("Введите целые a и b: ")

fmt.Scan(&a, &b)

serSum := findSeriesSum(a, b)

if serSum == -1 {

fmt.Println("Ряд расходится (бесконечная сумма)")

} else if !isPossiblyRational(serSum) {

fmt.Printf("Сумма — иррациональное число: %f\n", serSum)

}

}

Тестирование работы программы (Рисунок 11, Рисунок 12)

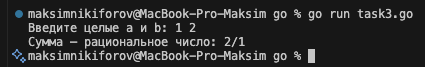


Рисунок 11 – Тестирование на сходящемся к рациональному числу ряде

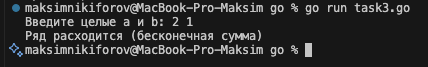


Рисунок 12 – Тестирование программы на расходящемся ряде

## Задание 4

### Реализация на C++.

#include <iostream>

#include <algorithm>

#include <climits>

#include <vector>

using namespace std;

bool findWinner(int n, int m, const vector<int> &nums) {

// dp[i] — макс. разница очков (Павел - Вика),

vector<int> dp(n + 1, INT\_MIN);

dp[n] = 0; // если все числа уже удалены, разница очков = 0

// идем от обратного т.к результат предречен

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

int current\_sum = 0; // сумма чисел, которые мы возьмём на этом шаге

// Пробуем взять от 1 до m чисел если они остались

for (int k = 1; k <= m && i + k <= n; ++k) {

current\_sum += nums[i + k - 1];

// dp[i + k] — это разница очков, которую получит соперник после нас

// Т.е. забирает current\_sum, потом соперник играет с позиции i + k

dp[i] = max(dp[i], current\_sum - dp[i + k]);

}

}

return dp[0] > 0 ? 1 : 0;

}

int main() {

int n, m;

cout << "Введите n (5 ≤ n ≤ 50000) и m (4 ≤ m ≤ 100): ";

cin >> n >> m;

vector<int> nums(n);

cout << "Последовательность: ";

vector<int> nums(n);

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

cin >> nums[i];

}

cout << findWinner(n, m, nums);

return 0;

}

Тестирование работы программы (Рисунок 13).

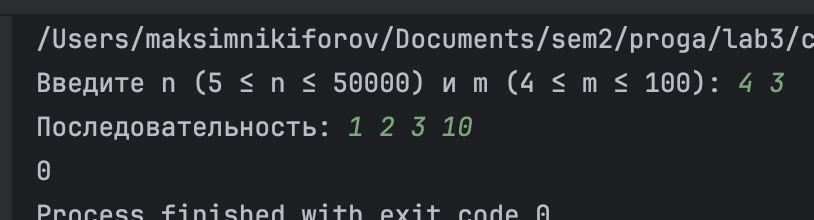


Рисунок 13 – тестирование работы программы.

### Реализация на Go.

package main

import (

"fmt"

"math"

)

func max(a, b int) int {

if a > b {

return a

}

return b

}

func findWinner(n, m int, nums []int) bool {

dp := make([]int, n+1)

for i := range dp {

dp[i] = math.MinInt

}

dp[n] = 0

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

currSum := 0

for k := 1; k <= m && i+k <= n; k++ {

currSum += nums[i+k-1]

dp[i] = max(dp[i], currSum-dp[i+k])

}

}

return dp[0] > 0

}

func main() {

var n, m int

fmt.Print("Введите n (5 ≤ n ≤ 50000) и m (4 ≤ m ≤ 100): ")

fmt.Scan(&n, &m)

nums := make([]int, n)

fmt.Print("Последовательность: ")

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

fmt.Scan(&nums[i])

}

if findWinner(n, m, nums) {

fmt.Println("1")

} else {

fmt.Println("0")

}

}

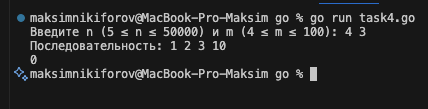
Тестирование работы программы (Рисунок 14).

Рисунок 14 – тестирование работы программы

# Вывод

В ходе лабораторной работы были достигнуты поставленные цели и выполнены задачи, такие как изучение циклических алгоритмов, операторов цикла, освоено программирование циклического вычислительного процесса. Были решены задачи по индивидуальным заданиям