МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра защиты информации

**

**ОТЧЁТ**

**по лабораторной работе №4**

**«**Последовательные одномерные контейнеры**»**

**по дисциплине: «***Программирование***»**

Выполнил:Проверил:

Студент гр. «АБс-324», «АВТФ» *Ассистент кафедры ЗИ*

*Клаус В. А. Исаев Г. А.*

«\_\_» \_\_\_\_\_\_ 2024 г.«\_\_\_» \_\_\_\_\_\_ 2024 г.

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

(подпись) (подпись)

Новосибирск 2024

**Цели и задачи работы:** изучение алгоритмов формирования и обработки одномерных массивов и последовательных контейнеров, программирование и отладка программ формирования и обработки массивов.

**Задание к работе:** Написать программу решения задачи в соответствии с индивидуальным вариантом.

**Задания:**

1. Отделите корни заданного уравнения, согласно варианту из таблицы, и уточните их одним из методов с точностью до ε=10-4 . Решить уравнения методом половинного деления, методом Ньютона и методом простых итераций. **Вариант 10 (2𝑥 + 𝑡𝑔(𝑥) = 0).**
2. 2.1. Напишите программу, в которой определен массив из 𝑛 чисел (𝑛 ≥ 10) и инициализирован случайными числами из диапазона [10, 100].

2.2. Найдите два наименьших элемента массива и вычислите сумму остатков получаемых от деления элементов массива из пункта 1 на минимальный элемент.

2.3. Определите массив. Найдите самую длинную возрастающую последовательность чисел в массиве. Определите и инициализируйте новый массив найденной последовательностью.

2.4. Вывести на экран массив длиной 𝑁 × 8. Первый элемент массива равен 2000. Необходимо выводить элементы строками по 8 элементов.

**Вариант 10. Алгоритм вариант 10 (random\_device).**

1. В соответствии с вариантом необходимо описать распределение генератора, с помощью этого генератора заполнить 3 массива размером 50, 100 и 1000 соответственно числами от 1 до 100. Использовать критерий хи-квадрат для проверки гипотезы о нормальном распределении выборки. Проанализировать значение критерия хи-квадрат и вывести:

1) результат проверки гипотезы;

2) ожидаемое и реальное математическое ожидание.

**Вариант 10. Алгоритм вариант 10 (random\_device).**

1. Реализовать иру «Предать или сотрудничать» и реализовать 3 алгоритма поведения в игре.
2. Генерация псевдослучайных чисел. **Вариант 5 (RC4).**

**Задание 1**

**C++:**

**// 2𝑥 + 𝑡𝑔(𝑥) = 0**

**#include <iostream>**

**#include <math.h>**

**#include <iomanip>**

**using namespace std;**

**// Метод Ньютона**

**double f(double x) { // значение функции**

**return 2\*x + tan(x);**

**}**

**double fsh(double x) { // значение производой функции**

**return 2 + 1/(cos(x)\*cos(x));**

**}**

**double newton(double x0, double eps) {**

**double x1 = x0 - f(x0) / fsh(x0);**

**int iternum = 1;**

**while (fabs(x1 - x0) > eps) {**

**cout << iternum << setw(15) << x0 << setw(15) << x1 << setw(15) << x1 - x0 << endl;**

**x0 = x1;**

**x1 = x0 - f(x0) / fsh(x0);**

**iternum++;**

**}**

**return x1;**

**}**

**// Метод простых итераций**

**double phi(double x) { // итерационная функция**

**return -0.5 \* tan(x);**

**}**

**double SimpleIterations(double x0, double eps) {**

**double x1 = phi(x0);**

**int iternum = 1;**

**while (fabs(x1 - x0) > eps) {**

**cout << iternum << setw(15) << x0 << setw(15) << x1 << setw(15) << x1 - x0 << endl;**

**x0 = x1;**

**x1 = phi(x0);**

**iternum++;**

**}**

**return x1;**

**}**

**// Метод половинного деления**

**double bisection(double a, double b, double eps) {**

**double c = 0.0;**

**int iternum = 1;**

**while (fabs(b - a) > eps) {**

**cout << iternum << setw(15) << a << setw(15) << b << setw(15) << b - a << endl;**

**c = (a + b) / 2;**

**if (f(a) \* f(c) < 0) {**

**b = c;**

**} else {**

**a = c;**

**}**

**iternum++;**

**}**

**return c;**

**}**

**int main() {**

**setlocale(LC\_ALL, "Rus");**

**// Метод Ньютона**

**cout << "Метод Ньютона:" << endl << "N" << setw(15) << "xn" << setw(15) << "xn+1" << setw(15) << "xn+1 - xn" << endl;**

**double x0 = 1.5; // начальное приближение**

**double eps = 0.001; // точность решения**

**double x = newton(x0, eps);**

**cout << "Корень: " << x << endl << endl;**

**// Метод простых итераций**

**cout << "Метод простых итераций:" << endl << "N" << setw(15) << "xn" << setw(15) << "xn+1" << setw(15) << "xn+1 - xn" << endl;**

**x0 = 1.0; // начальное приближение**

**eps = 0.001; // точность решение**

**x = SimpleIterations(x0, eps);**

**cout << "Корень: " << x << endl << endl;**

**// Метод половинного деления**

**cout << "Метод половинного деления:" << endl << "N" << setw(15) << "an" << setw(15) << "bn" << setw(15) << "bn - an" << endl;**

**double a = -2.0; // начальный интервал**

**double b = 2.0;**

**eps = 0.001; // точность решения**

**x = bisection(a, b, eps);**

**cout << "Корень: " << x << endl << endl;**

**return 0;**

**}**

**Input: Метод Ньютона:**

**N xn xn+1 xn+1 - xn**

**1 1.5 1.41528 -0.0847234**

**2 1.41528 1.20446 -0.210819**

**3 1.20446 0.692358 -0.5121**

**4 0.692358 0.0919857 -0.600372**

**5 0.0919857 0.000173645 -0.0918121**

**Корень: 1.16353e-12**

**Метод простых итераций:**

**N xn xn+1 xn+1 - xn**

**1 1 -0.778704 -1.7787**

**2 -0.778704 0.49335 1.27205**

**3 0.49335 -0.26885 -0.7622**

**4 -0.26885 0.13776 0.40661**

**5 0.13776 -0.0693191 -0.207079**

**6 -0.0693191 0.0347151 0.104034**

**7 0.0347151 -0.0173646 -0.0520797**

**8 -0.0173646 0.00868315 0.0260477**

**9 0.00868315 -0.00434168 -0.0130248**

**10 -0.00434168 0.00217086 0.00651254**

**11 0.00217086 -0.00108543 -0.00325628**

**12 -0.00108543 0.000542715 0.00162814**

**Корень: -0.000271357**

**Метод половинного деления:**

**N an bn bn - an**

**1 -2 2 4**

**2 0 2 2**

**3 1 2 1**

**4 1.5 2 0.5**

**5 1.5 1.75 0.25**

**6 1.5 1.625 0.125**

**7 1.5625 1.625 0.0625**

**8 1.5625 1.59375 0.03125**

**9 1.5625 1.57812 0.015625**

**10 1.57031 1.57812 0.0078125**

**11 1.57031 1.57422 0.00390625**

**12 1.57031 1.57227 0.00195312**

**Корень: 1.57129**

**Python:**

**import math**

**# Метод Ньютона**

**def f(x): return 2\*x + math.tan(x) # значение ф-ии**

**def fsh(x): return 2 + 1/(math.cos(x)\*math.cos(x)) # значение производной ф-ии**

**def newton(x0, eps):**

**x1 = x0 - f(x0) / fsh(x0)**

**iternum = 1**

**while abs(x1 - x0) > eps:**

**print(iternum, "{:15.6f}".format(x0), "{:15.6f}".format(x1), "{:15.6f}".format(x1 - x0))**

**x0 = x1**

**x1 = x0 - f(x0) / fsh(x0)**

**iternum += 1**

**return x1**

**# Метод простых итераций**

**def phi(x): return -0.5 \* math.tan(x) # итерационная ф-ия**

**def SimpleIterations(x0, eps):**

**x1 = phi(x0)**

**iternum = 1**

**while abs(x1 - x0) > eps:**

**print(iternum, "{:15.6f}".format(x0), "{:15.6f}".format(x1), "{:15.6f}".format(x1 - x0))**

**x0 = x1**

**x1 = phi(x0)**

**iternum += 1**

**return x1**

**def bisection(a, b, eps):**

**c = 0.0**

**iternum = 1**

**while abs(b - a) > eps:**

**print(iternum, "{:15.6f}".format(a), "{:15.6f}".format(b), "{:15.6f}".format(b - a))**

**c = (a + b) / 2**

**if f(a) \* f(c) < 0: b = c**

**else: a = c**

**iternum += 1**

**return c**

**# Метод Ньютона**

**print("Метод Ньютона:")**

**print("N".ljust(15) + "xn".ljust(15) + "xn+1".ljust(15) + "xn+1 - xn")**

**x0 = 1.5 # начальное приближение**

**eps = 0.001 # точность решения**

**x = newton(x0, eps)**

**print("Корень:", x, "\n")**

**# Метод простых итераций**

**print("Метод простых итераций:")**

**print("N".ljust(15) + "xn".ljust(15) + "xn+1".ljust(15) + "xn+1 - xn")**

**x0 = 1.0 # начальное приближение**

**eps = 0.001 # точность решения**

**x = SimpleIterations(x0, eps)**

**print("Корень:", x, "\n")**

**# Метод половинного деления**

**print("Метод половинного деления:")**

**print("N".ljust(15) + "an".ljust(15) + "bn".ljust(15) + "bn - an")**

**a = -2.0 # начальный интервал**

**b = 2.0**

**eps = 0.001 # точность решения**

**x = bisection(a, b, eps)**

**print("Корень:", x, "\n")**

**Input: Метод Ньютона:**

**N xn xn+1 xn+1 - xn**

**1 1.500000 1.415277 -0.084723**

**2 1.415277 1.204458 -0.210819**

**3 1.204458 0.692358 -0.512100**

**4 0.692358 0.091986 -0.600372**

**5 0.091986 0.000174 -0.091812**

**Корень: 1.1635312396722497e-12**

**Метод простых итераций:**

**N xn xn+1 xn+1 - xn**

**1 1.000000 -0.778704 -1.778704**

**2 -0.778704 0.493350 1.272054**

**3 0.493350 -0.268850 -0.762200**

**4 -0.268850 0.137760 0.406610**

**5 0.137760 -0.069319 -0.207079**

**6 -0.069319 0.034715 0.104034**

**7 0.034715 -0.017365 -0.052080**

**8 -0.017365 0.008683 0.026048**

**9 0.008683 -0.004342 -0.013025**

**10 -0.004342 0.002171 0.006513**

**11 0.002171 -0.001085 -0.003256**

**12 -0.001085 0.000543 0.001628**

**Корень: -0.00027135745705895184**

**Метод половинного деления:**

**N an bn bn - an**

**1 -2.000000 2.000000 4.000000**

**2 0.000000 2.000000 2.000000**

**3 1.000000 2.000000 1.000000**

**4 1.500000 2.000000 0.500000**

**5 1.500000 1.750000 0.250000**

**6 1.500000 1.625000 0.125000**

**7 1.562500 1.625000 0.062500**

**8 1.562500 1.593750 0.031250**

**9 1.562500 1.578125 0.015625**

**10 1.570312 1.578125 0.007812**

**11 1.570312 1.574219 0.003906**

**12 1.570312 1.572266 0.001953**

**Корень: 1.5712890625**

**Задание 2**

**С++:**

**// Задание вариант 10 (Вариант алгоритма 10 (random\_device))**

**#include <iostream>**

**#include <vector>**

**#include <random>**

**using namespace std;**

**vector<int> FindLongSeq(vector<int> nums) { // Самая длинная последовательность массива**

**vector<int> LongSeq;**

**vector<int> CurrentSeq;**

**for(int i = 0; i < nums.size(); ++i) {**

**if (CurrentSeq.empty() || nums[i] > CurrentSeq.back()) {**

**CurrentSeq.push\_back(nums[i]);**

**} else {**

**if (CurrentSeq.size() > LongSeq.size()) {**

**LongSeq = CurrentSeq;**

**}**

**CurrentSeq.clear();**

**CurrentSeq.push\_back(nums[i]);**

**}**

**}**

**if (CurrentSeq.size() > LongSeq.size()) {**

**LongSeq = CurrentSeq;**

**}**

**return LongSeq;**

**}**

**int main(){**

**srand(time(0));**

**setlocale(LC\_ALL, "Rus");**

**random\_device generator;**

**// Блок 1**

**int n = 0;**

**while (n > 10 || n <= 0) {**

**cout << "1. Введите размерность массива рандомных чисел(не больше 10): ";**

**cin >> n;**

**}**

**vector<int> nums;**

**int min1=1000, min2=1000;**

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

**int randnums = 10 + generator() % 90; // Генерирую число, записываю его в контейнер**

**nums.push\_back(randnums);**

**if (randnums < min1) { // нахожу 2 минимальных**

**min2 = min1;**

**min1 = randnums;**

**} else if (randnums < min2 && randnums != min1) {**

**min2 = randnums;**

**}**

**}**

**cout << "Рандомные числа: ";**

**for (int i : nums) {**

**cout << i << " ";**

**}**

**cout << endl << endl;**

**// Блок 2**

**cout << "2. Минимальные числа: " << min1 << " " << min2 << endl;**

**int sumost = 0;**

**for (int i : nums) { // считаю сумма отсаткой при делении элементов массива на минимальное число**

**sumost += (i % min1);**

**}**

**cout << "Сумма остатков: " << sumost << endl << endl;**

**// Блок 3**

**n = 0;**

**while (n > 10 || n <= 0) {**

**cout << "3. Введите размерность массива чисел(не больше 10): ";**

**cin >> n;**

**}**

**vector<int> nums2(n);**

**cout << "Введите массив чисел: ";**

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

**cin >> nums2[i];**

**}**

**vector<int> LongSeq = FindLongSeq(nums2); // ф-ия нахождения самой длинной последовательности массива**

**cout << "Самая длинная последовательность массива: ";**

**for (int i : LongSeq) {**

**cout << i << " ";**

**}**

**cout << endl << endl;**

**// Блок 4**

**n = 0;**

**while(n <= 0 || n > 10) {**

**cout << "4. Введите кол-во столбцов в массиве(Не больше 10): ";**

**cin >> n;**

**}**

**cout << "Массив: " << endl;**

**n \*= 8;**

**int count = 2;**

**vector<int> nums3(n);**

**nums3[0] = 2000;**

**cout << nums3[0] << " ";**

**for (int i = 1; i < n; ++i) {**

**nums3[i] = nums3[i-1] + 10;**

**cout << nums3[i] << " ";**

**if (count % 8 == 0) {**

**cout << endl;**

**}**

**count++;**

**}**

**return 0;**

**}**

**Input: 1. Введите размерность массива рандомных чисел(не больше 10): 5**

**Рандомные числа: 83 12 60 19 97**

**2. Минимальные числа: 12 19**

**Сумма остатков: 19**

**3. Введите размерность массива чисел(не больше 10): 5**

**Введите массив чисел: 1 2 3 1 2**

**Самая длинная последовательность массива: 1 2 3**

**4. Введите кол-во столбцов в массиве(Не больше 10): 5**

**Массив:**

**2000 2010 2020 2030 2040 2050 2060 2070**

**2080 2090 2100 2110 2120 2130 2140 2150**

**2160 2170 2180 2190 2200 2210 2220 2230**

**2240 2250 2260 2270 2280 2290 2300 2310**

**2320 2330 2340 2350 2360 2370 2380 2390**

**Python:**

**import random**

**def find\_long\_seq(nums): # Самая длинная последовательность массива**

**long\_seq = []**

**current\_seq = []**

**for i in range(len(nums)):**

**if not current\_seq or nums[i] > current\_seq[-1]:**

**current\_seq.append(nums[i])**

**else:**

**if len(current\_seq) > len(long\_seq):**

**long\_seq = current\_seq**

**current\_seq = [nums[i]]**

**if len(current\_seq) > len(long\_seq):**

**long\_seq = current\_seq**

**return long\_seq**

**# Блок 1**

**n = 0**

**while n > 10 or n <= 0:**

**n = int(input("1. Введите размерность массива рандомных чисел(не больше 10): "))**

**nums = []**

**min1 = 1000**

**min2 = 1000**

**for i in range(n):**

**randnums = 10 + random.randint(0, 100) # Генерирую число, записываю его в контейнер**

**nums.append(randnums)**

**if randnums < min1: # нахожу 2 минимальных**

**min2 = min1**

**min1 = randnums**

**elif randnums < min2 and randnums != min1:**

**min2 = randnums**

**print("Рандомные числа:", end=" ")**

**for i in nums:**

**print(i, end=" ")**

**print("\n")**

**# Блок 2**

**print("2. Минимальные числа:", min1, min2)**

**sumost = 0**

**for i in nums: # считаю сумма отсатков при делении элементов массива на минимальное число**

**sumost += (i % min1)**

**print("Сумма остатков:", sumost, "\n")**

**# Блок 3**

**n = 0**

**while n > 10 or n <= 0:**

**print("3. Введите размерность массива чисел(не больше 10): ", end="")**

**n = int(input())**

**nums2 = [0] \* n**

**print("Введите массив чисел: ")**

**for i in range(n):**

**nums2[i] = int(input())**

**LongSeq = find\_long\_seq(nums2) # функция нахождения самой длинной последовательности массива**

**print("Самая длинная последовательность массива: ", end="")**

**for i in LongSeq:**

**print(i, end=" ")**

**print("\n\n")**

**# Блок 4**

**n = 0**

**while n <= 0 or n > 10:**

**print("4. Введите кол-во столбцов в массиве(Не больше 10): ", end="")**

**n = int(input())**

**print("Массив: ")**

**n \*= 8**

**count = 2**

**nums3 = [0] \* n**

**nums3[0] = 2000**

**print(nums3[0], end=" ")**

**for i in range(1, n):**

**nums3[i] = nums3[i - 1] + 10**

**print(nums3[i], end=" ")**

**if count % 8 == 0:**

**print()**

**count += 1**

**Input: 1. Введите размерность массива рандомных чисел(не больше 10): 5**

**Рандомные числа: 13 10 90 78 100**

**2. Минимальные числа: 10 13**

**Сумма остатков: 11**

**3. Введите размерность массива чисел(не больше 10): 5**

**Введите массив чисел:**

**1**

**2**

**3**

**1**

**2**

**Самая длинная последовательность массива: 1 2 3**

**4. Введите кол-во столбцов в массиве(Не больше 10): 5**

**Массив:**

**2000 2010 2020 2030 2040 2050 2060 2070**

**2080 2090 2100 2110 2120 2130 2140 2150**

**2160 2170 2180 2190 2200 2210 2220 2230**

**2240 2250 2260 2270 2280 2290 2300 2310**

**2320 2330 2340 2350 2360 2370 2380 2390**

**Задание 3**

**С++:**

**#include <iostream>**

**#include <random>**

**#include <vector>**

**#include <math.h>**

**using namespace std;**

**int main(){**

**setlocale(LC\_ALL, "Rus");**

**srand(time(0));**

**random\_device generator;**

**// Генерирую числа для каждого массива**

**vector<int> arr50(50);**

**vector<int> arr100(100);**

**vector<int> arr1000(1000);**

**for (int i = 0; i < 50; ++i) {**

**arr50[i] = 10 + generator() % 90;**

**}**

**for (int i = 0; i < 100; ++i) {**

**arr100[i] = 10 + generator() % 90;**

**}**

**for (int i = 0; i < 1000; ++i) {**

**arr1000[i] = 10 + generator() % 90;**

**}**

**// Проверка гипотезы о нормальном распределении с использованием критерия хи-квадрат**

**// Для начала определим ожидаемое количество попаданий в каждый интервал.**

**// Поскольку у нас 100 интервалов (числа от 1 до 100), ожидаемое количество попаданий в каждый интервал будет равно общему количеству элементов делённому на количество интервалов.**

**int expected\_count = 50; // Для выборок размером 50, 100 и 1000 ожидаемое количество элементов в каждом интервале будет одинаковым**

**// Рассчитываем значения критериев xи-квадрата для каждой выборки**

**double chi\_square\_50 = 0.0;**

**double chi\_square\_100 = 0.0;**

**double chi\_square\_1000 = 0.0;**

**for (int i = 1; i <= 100; ++i) {**

**// Для каждого интервала подсчитываем количество элементов из выборки, которые попадают в данный интервал**

**int count\_50 = 0;**

**int count\_100 = 0;**

**int count\_1000 = 0;**

**for (int j = 0; j < 50; ++j) {**

**if (arr50[j] == i) {**

**count\_50++;**

**}**

**}**

**for (int j = 0; j < 100; ++j) {**

**if (arr100[j] == i) {**

**count\_100++;**

**}**

**}**

**for (int j = 0; j < 1000; ++j) {**

**if (arr1000[j] == i) {**

**count\_1000++;**

**}**

**}**

**// Рассчитываем значение критерия хи-квадрат для каждой выборки**

**chi\_square\_50 += pow(count\_50 - expected\_count, 2) / expected\_count;**

**chi\_square\_100 += pow(count\_100 - expected\_count, 2) / expected\_count;**

**chi\_square\_1000 += pow(count\_1000 - expected\_count, 2) / expected\_count;**

**}**

**cout << "Критерий Хи-квадрат для выборки размером 50: " << chi\_square\_50 << endl;**

**cout << "Критерий Хи-квадрат для выборки размером 100: " << chi\_square\_100 << endl;**

**cout << "Критерий Хи-квадрат для выборки размером 1000: " << chi\_square\_1000 << endl;**

**// Вывод результата проверки гипотезы**

**// Для простоты проверяем, является ли значение критерия меньше критического значения, например, 124.342 (значение выбрано произвольно)**

**double critical\_value = 124.342;**

**if (chi\_square\_50 < critical\_value && chi\_square\_100 < critical\_value && chi\_square\_1000 < critical\_value) {**

**cout << "Гипотеза о нормальном распределении не отвергается." << endl;**

**} else {**

**cout << "Гипотеза о нормальном распределении отвергается." << endl;**

**}**

**// Вывод ожидаемого и реального математического ожидания (для простоты, они одинаковы для всех выборок)**

**cout << "Ожидаемое количество попаданий в каждый интервал: " << expected\_count << endl;**

**cout << "Реальное количество попаданий в каждый интервал: " << expected\_count << endl;**

**return 0;**

**}**

**Input: Критерий Хи-квадрат для выборки размером 50: 4901.2**

**Критерий Хи-квадрат для выборки размером 100: 4804.44**

**Критерий Хи-квадрат для выборки размером 1000: 3239.84**

**Гипотеза о нормальном распределении отвергается.**

**Ожидаемое количество попаданий в каждый интервал: 50**

**Реальное количество попаданий в каждый интервал: 50**

**Задание 4**

**С++:**

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**bool sotr(int round\_nubmber, vector<bool> self\_choices, vector<bool> enemy\_choices) { // Всегда сотрудничает**

**return true;**

**}**

**bool pred(int round\_number, vector<bool> self\_choices, vector<bool> enemy\_choices) { // Всегда предает**

**return false;**

**}**

**bool notrepeat(int round\_number, vector<bool> self\_choices, vector<bool> enemy\_choices) { // Сначала сотрудничает, а далее не повторяет выбор соперника за предыдущих ход**

**if (round\_number == 0) return true; // первый раунд - сотрудничает**

**else if (enemy\_choices[round\_number - 1] == true) return false; // не повторяет предыдущий ход соперника**

**else return true;**

**}**

**int main() {**

**setlocale(LC\_ALL, "Rus");**

**srand(time(0));**

**// Игра 1(Мы всегда сотрудничаем, противник всегда не повторяет)**

**int rounds = 100 + rand() % 101;**

**vector<bool> self\_choices; // Свои решения**

**vector<bool> enemy\_choices; // решения противника**

**for (int i = 0; i < rounds; i++) {**

**bool a\_choice = sotr(i, self\_choices, enemy\_choices);**

**bool b\_choice = notrepeat(i, enemy\_choices, self\_choices);**

**if (a\_choice == true && b\_choice == true) { // оба сотрудничают**

**self\_choices.push\_back(true);**

**enemy\_choices.push\_back(true);**

**} else if (a\_choice == false && b\_choice == false) { // оба предают**

**self\_choices.push\_back(false);**

**enemy\_choices.push\_back(false);**

**} else if (a\_choice == true && b\_choice == false) { // a сотрудничает, b предает**

**self\_choices.push\_back(true);**

**enemy\_choices.push\_back(false);**

**} else { // a предает, b сотрудничает**

**self\_choices.push\_back(false);**

**enemy\_choices.push\_back(true);**

**}**

**}**

**// Подсчет очков**

**int score\_a = 0;**

**int score\_b = 0;**

**for (int i = 0; i < rounds; i++) {**

**if (self\_choices[i] == true && enemy\_choices[i] == true) {**

**score\_a += 24;**

**score\_b += 24;**

**} else if (self\_choices[i] == false && enemy\_choices[i] == false) {**

**score\_a += 4;**

**score\_b += 4;**

**} else if (self\_choices[i] == true && enemy\_choices[i] == false) {**

**score\_a += 0;**

**score\_b += 20;**

**} else {**

**score\_a += 20;**

**score\_b += 0;**

**}**

**}**

**cout << "Игра 1.\nКол-во раундов: " << rounds << "\nАлгоритм A: " << score\_a << "\nАлгоритм B: " << score\_b << endl;**

**// Игра 2(Мы всегда предаём, противник всегда не повторяет)**

**rounds = 100 + rand() % 101;**

**self\_choices.clear(); // Свои решения**

**enemy\_choices.clear(); // решения противника**

**for (int i = 0; i < rounds; i++) {**

**bool a\_choice = pred(i, self\_choices, enemy\_choices);**

**bool b\_choice = notrepeat(i, enemy\_choices, self\_choices);**

**if (a\_choice == true && b\_choice == true) { // оба сотрудничают**

**self\_choices.push\_back(true);**

**enemy\_choices.push\_back(true);**

**} else if (a\_choice == false && b\_choice == false) { // оба предают**

**self\_choices.push\_back(false);**

**enemy\_choices.push\_back(false);**

**} else if (a\_choice == true && b\_choice == false) { // a сотрудничает, b предает**

**self\_choices.push\_back(true);**

**enemy\_choices.push\_back(false);**

**} else { // a предает, b сотрудничает**

**self\_choices.push\_back(false);**

**enemy\_choices.push\_back(true);**

**}**

**}**

**// Подсчет очков**

**score\_a = 0;**

**score\_b = 0;**

**for (int i = 0; i < rounds; i++) {**

**if (self\_choices[i] == true && enemy\_choices[i] == true) {**

**score\_a += 24;**

**score\_b += 24;**

**} else if (self\_choices[i] == false && enemy\_choices[i] == false) {**

**score\_a += 4;**

**score\_b += 4;**

**} else if (self\_choices[i] == true && enemy\_choices[i] == false) {**

**score\_a += 0;**

**score\_b += 20;**

**} else {**

**score\_a += 20;**

**score\_b += 0;**

**}**

**}**

**cout << "Игра 2.\nКол-во раундов: " << rounds << "\nАлгоритм A: " << score\_a << "\nАлгоритм B: " << score\_b << endl;**

**// Игра 3(Мы всегда не повторяем, противник всегда не повторяет)**

**rounds = 100 + rand() % 101;**

**self\_choices.clear(); // Свои решения**

**enemy\_choices.clear(); // решения противника**

**for (int i = 0; i < rounds; i++) {**

**bool a\_choice = notrepeat(i, self\_choices, enemy\_choices);**

**bool b\_choice = notrepeat(i, enemy\_choices, self\_choices);**

**if (a\_choice == true && b\_choice == true) { // оба сотрудничают**

**self\_choices.push\_back(true);**

**enemy\_choices.push\_back(true);**

**} else if (a\_choice == false && b\_choice == false) { // оба предают**

**self\_choices.push\_back(false);**

**enemy\_choices.push\_back(false);**

**} else if (a\_choice == true && b\_choice == false) { // a сотрудничает, b предает**

**self\_choices.push\_back(true);**

**enemy\_choices.push\_back(false);**

**} else { // a предает, b сотрудничает**

**self\_choices.push\_back(false);**

**enemy\_choices.push\_back(true);**

**}**

**}**

**// Подсчет очков**

**score\_a = 0;**

**score\_b = 0;**

**for (int i = 0; i < rounds; i++) {**

**if (self\_choices[i] == true && enemy\_choices[i] == true) {**

**score\_a += 24;**

**score\_b += 24;**

**} else if (self\_choices[i] == false && enemy\_choices[i] == false) {**

**score\_a += 4;**

**score\_b += 4;**

**} else if (self\_choices[i] == true && enemy\_choices[i] == false) {**

**score\_a += 0;**

**score\_b += 20;**

**} else {**

**score\_a += 20;**

**score\_b += 0;**

**}**

**}**

**cout << "Игра 3.\nКол-во раундов: " << rounds << "\nАлгоритм A: " << score\_a << "\nАлгоритм B: " << score\_b << endl;**

**return 0;**

**}**

**Input: Игра 1.**

**Кол-во раундов: 124**

**Алгоритм A: 24**

**Алгоритм B: 2484**

**Игра 2.**

**Кол-во раундов: 152**

**Алгоритм A: 3040**

**Алгоритм B: 0**

**Игра 3.**

**Кол-во раундов: 176**

**Алгоритм A: 2464**

**Алгоритм B: 2464**

**Python:**

**import random**

**def sotr(round\_number, self\_choices, enemy\_choices): # Всегда сотрудничает**

**return True**

**def pred(round\_number, self\_choices, enemy\_choices): # Всегда предает**

**return False**

**def notrepeat(round\_number, self\_choices, enemy\_choices): # Сначала сотрудничает, а далее не повторяет выбор соперника за предыдущих ход**

**if round\_number == 0: return True # первый раунд - сотрудничает**

**elif enemy\_choices[round\_number - 1]: return False # не повторяет предыдущий ход соперника**

**else: return True**

**# Игра 1(Мы всегда сотрудничаем, противник всегда не повторяет)**

**rounds = random.randint(100, 200)**

**self\_choices = [] # свои решения**

**enemy\_choices = [] # решения противника**

**for i in range(rounds):**

**a\_choice = sotr(i, self\_choices, enemy\_choices)**

**b\_choice = notrepeat(i, enemy\_choices, self\_choices)**

**if a\_choice and b\_choice: # оба сотрудничают**

**self\_choices.append(True)**

**enemy\_choices.append(True)**

**elif not a\_choice and not b\_choice: # оба предают**

**self\_choices.append(False)**

**enemy\_choices.append(False)**

**elif a\_choice and not b\_choice: # А - сотрудничает, B - не сотрудничает**

**self\_choices.append(True)**

**enemy\_choices.append(False)**

**else: # A - не сотрудничает, B - сотрудничает**

**self\_choices.append(False)**

**enemy\_choices.append(True)**

**# Подсчет очков**

**score\_a = 0**

**score\_b = 0**

**for i in range(rounds):**

**if self\_choices[i] and enemy\_choices[i]:**

**score\_a += 24**

**score\_b += 24**

**elif not self\_choices[i] and not enemy\_choices[i]:**

**score\_a += 4**

**score\_b += 4**

**elif self\_choices[i] and not enemy\_choices[i]:**

**score\_a += 0**

**score\_b += 20**

**else:**

**score\_a += 20**

**score\_b += 0**

**print(f"Игра 1.\nКол-во раундов: {rounds}\nАлгоритм A: {score\_a}\nАлгоритм B: {score\_b}\n")**

**# Игра 2(Мы всегда предаём, противник всегда не повторяет)**

**rounds = random.randint(100, 200)**

**self\_choices = [] # свои решения**

**enemy\_choices = [] # решения противника**

**for i in range(rounds):**

**a\_choice = pred(i, self\_choices, enemy\_choices)**

**b\_choice = notrepeat(i, enemy\_choices, self\_choices)**

**if a\_choice and b\_choice:**

**self\_choices.append(True)**

**enemy\_choices.append(True)**

**elif not a\_choice and not b\_choice:**

**self\_choices.append(False)**

**enemy\_choices.append(False)**

**elif a\_choice and not b\_choice:**

**self\_choices.append(True)**

**enemy\_choices.append(False)**

**else:**

**self\_choices.append(False)**

**enemy\_choices.append(True)**

**# Подсчет очков**

**score\_a = 0**

**score\_b = 0**

**for i in range(rounds):**

**if self\_choices[i] and enemy\_choices[i]:**

**score\_a += 24**

**score\_b += 24**

**elif not self\_choices[i] and not enemy\_choices[i]:**

**score\_a += 4**

**score\_b += 4**

**elif self\_choices[i] and not enemy\_choices[i]:**

**score\_a += 0**

**score\_b += 20**

**else:**

**score\_a += 20**

**score\_b += 0**

**print(f"Игра 2.\nКол-во раундов: {rounds}\nАлгоритм A: {score\_a}\nАлгоритм B: {score\_b}\n")**

**# Игра 3(Мы всегда не повторяем, противник всегда не повторяет)**

**rounds = random.randint(100, 200)**

**self\_choices = [] # свои решения**

**enemy\_choices = [] # решения противника**

**for i in range(rounds):**

**a\_choice = notrepeat(i, self\_choices, enemy\_choices)**

**b\_choice = notrepeat(i, enemy\_choices, self\_choices)**

**if a\_choice and b\_choice:**

**self\_choices.append(True)**

**enemy\_choices.append(True)**

**elif not a\_choice and not b\_choice:**

**self\_choices.append(False)**

**enemy\_choices.append(False)**

**elif a\_choice and not b\_choice:**

**self\_choices.append(True)**

**enemy\_choices.append(False)**

**else:**

**self\_choices.append(False)**

**enemy\_choices.append(True)**

**# Подсчет очков**

**score\_a = 0**

**score\_b = 0**

**for i in range(rounds):**

**if self\_choices[i] and enemy\_choices[i]:**

**score\_a += 24**

**score\_b += 24**

**elif not self\_choices[i] and not enemy\_choices[i]:**

**score\_a += 4**

**score\_b += 4**

**elif self\_choices[i] and not enemy\_choices[i]:**

**score\_a += 0**

**score\_b += 20**

**else:**

**score\_a += 20**

**score\_b += 0**

**print(f"Игра 3.\nКол-во раундов: {rounds}\nАлгоритм A: {score\_a}\nАлгоритм B: {score\_b}\n")**

**Input: Игра 1.**

**Кол-во раундов: 168**

**Алгоритм A: 24**

**Алгоритм B: 3364**

**Игра 2.**

**Кол-во раундов: 163**

**Алгоритм A: 3260**

**Алгоритм B: 0**

**Игра 3.**

**Кол-во раундов: 104**

**Алгоритм A: 1456**

**Алгоритм B: 1456**

**Задание 5**

**С++:**

**// вариант 5(RC4)**

**#include <iostream>**

**#include <vector>**

**#include <math.h>**

**using namespace std;**

**vector<int> RC4(int n, int GeneratingOutput) {**

**// Инициализация таблиц**

**vector<int> S; // Таблица S**

**for (int i = 0; i < pow(2, n); i++) { // таблица замен S**

**S.push\_back(i);**

**}**

**vector<int> K; // Таблица K**

**int k = 1; // Значение ключа**

**for (int i = 0; i < pow(2, n); i++) { // Таблица K с ключом от 1 до 6**

**if (k != 6) { // Записываем в таблицу K значения от 1 до 6**

**K.push\_back(k);**

**k++;**

**} else {**

**K.push\_back(k);**

**k = 1;**

**}**

**}**

**int j = 0;**

**for (int i = 0; i < pow(2, n); i++) { // перестановка**

**j = (j + S[i] + K[i]) % (int(pow(2, n)));**

**swap(S[i], S[j]);**

**}**

**// Алгоритм псевдослучайной генерации**

**vector<int> Z; // Рандомные числа**

**int i = 0, t = 0, z = 0;**

**j = 0;**

**while (GeneratingOutput != 0) {**

**i = (i + 1) % (int(pow(2, n)));**

**j = (j + S[i]) % (int(pow(2, n)));**

**swap(S[i], S[j]);**

**t = (S[i] + S[j]) % (int(pow(2, n)));**

**z = S[t];**

**Z.push\_back(z);**

**GeneratingOutput--;**

**}**

**return Z;**

**}**

**int main(){**

**setlocale(LC\_ALL, "Rus");**

**int n;**

**cout << "Введите число n: ";**

**cin >> n;**

**int k;**

**cout << "Введите необходимое кол-во рандомных чисел: ";**

**cin >> k;**

**cout << "Рандомные числа:" << endl;**

**vector<int> randomnums = RC4(n, k);**

**for (int i : randomnums) {**

**cout << i << " ";**

**}**

**return 0;**

**}**

**Input: Введите число n: 4**

**Введите необходимое кол-во рандомных чисел: 5**

**Рандомные числа:**

**2 4 10 15 3**

**Вывод:**

В ходе выполнения лабораторной работы №4 я познакомился с новыми методами генерации рандомных чисел и самостоятельно реализовал один из них - RC4, который пригодится мне в дальнейшем изучении криптографии.

**Ссылка на репозиторий:** [нажмите](https://github.com/JamesSevil/laba4)