МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХ™НИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра защиты информации

ЛАБОРАТОРНАЯ РАБОТА №4

«Последовательные одномерные контейнеры»
по дисциплине: «*Программирование*»

Выполнил:	Проверил:
Студент гр. «АБс-323», «АВТФ»	*Ассистент кафедры ЗИ*
Пушкарев Виталий Иванович	*Исаев Глеб Андреевич*
«27» _____05_____ 2024г	«___» _____ 2024г
_____	_____
(подпись)	(подпись)

Новосибирск 2024

Оглавление

Цели и задачи работы. 3
Задание к работе 3
Индивидуальное задание 3
Задание 1: 5
Задание 2: 10
Задание 3: 17
Задание 4: 24
Задание 5: 28

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.

Индивидуальное задание:

Задание 1. № 5. 2x+cos(x)=0.

*Задание 2. №*6

1. Напишите программу, в которой определен массив $arr1$ из n чисел ($n \geq 10$) и инициализирован целыми случайными числами из диапазона [0, 100].
2. Найдите сумму элементов массива $arr1$ между наибольшим и наименьшим элементами.
3. Определите и инициализируйте новый массив $arr2$ элементами, расположенными между наибольшим и наименьшим элементом массива $arr1$, массив $arr2$ должен иметь такую же размерность, что и $arr1$, поэтому дополните свободные места случайными элементами из массива $arr1$.
4. Определите и инициализируйте массив, состоящий из случайных символов. Напишите программу, сортирующую в порядке возрастания элементы на чётных местах.
5. Определите и инициализируйте массив, состоящий из случайных целых чисел, входящих в диапазон [100, 900]. Отсортируйте массив по возрастанию. Определите новый массив. Поменяйте цифры в элементах исходного массива в случайном порядке. Если число окажется больше исходного, то добавьте его в новый массив, если нет – вместо модифицированного числа запишите в новый массив 0.

*Задание 3. №*1. Реализовать вычисление хи-квадрат для minstd_rand и проверить является ли выборка этого генератора нормальным распределением.

Задание 4. Требуется реализовать игру «Предать или сотрудничать» и реализовать 3 алгоритма поведения в игре. Игра состоит из случайного кол-ва раундов от 100 до 200 (итоговое кол-во раундов при каждом запуске игры генерируется случайно). На протяжении игровой сессии сражаются 2 алгоритма. В каждом раунде каждый алгоритм выбирает, либо сотрудничество, либо предательство. Если алгоритм А выбирает предательство и алгоритм Б выбирает предательство они получают по 4 очка. Если алгоритм А выбирает сотрудничество, а алгоритм Б выбирает предательство - алгоритм А получает 0 очков, а алгоритм Б получает 20 очков. Если оба алгоритма выбирают сотрудничество оба получают 24 очка. Каждому алгоритму в каждом раунде известны результаты всех предыдуще

раундов текущей игровой сессии, на основе этих данных алгоритм может выбирать будет он сотрудничать или предаст. Каждый алгоритм должен представлять из себя функцию с сигнатурой: boolean func(int32 round_number, array[boolean] self_choices, array[boolean] enemy_choices) round_number – номер текущего раунда self_choices – массив булевых значений, содержит информацию о собственных выборов (предать или сотрудничать) за все предыдущие раунды enemy_choices - массив булевых значений, содержит информацию о выборов (предать или сотрудничать) противника за все предыдущие раунды true – сотрудннчество false – предательство

Задание 5. № 1.

Реализовать линейный конгурэнтный генератор.

Задание 1:
C++

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;

void InputTable(double LeftPoint, double RightPoint, int N)
{
    if (N == 1)
    {
        cout << "N";
        cout << "\t     a ";
        cout << "\t\t\t     b ";
        cout << "\t\t\t     b-a ";
        cout << endl;
        cout << "-------------------------------------------------------------------------
------------";
        cout << endl;
    }
    cout << N << " |";
    cout << "\t" << setw(10) << LeftPoint;
    cout << "\t\t" << setw(10) << RightPoint;
    cout << "\t\t" << setw(12) << RightPoint - LeftPoint;
    cout << endl;
}
void InputNewtonMethod(double LeftPoint, double RightPoint, int N)
{
    if (N == 1)
    {
        cout << "N";
        cout << "\t     Xn ";
        cout << "\t\t\t     Xn+1 ";
        cout << "\t\t\t     Xn+1-Xn ";
        cout << endl;
        cout << "-------------------------------------------------------------------------
------------";
        cout << endl;
    }
    cout << N << " |";
    cout << "\t" << setw(10) << LeftPoint;
    cout << "\t\t" << setw(10) << RightPoint;
    cout << "\t\t" << setw(12) << RightPoint - LeftPoint;
    cout << endl;
}
double function(double x) {
    return 2 * x + cos(x); // Заменить функцией, корни которой мы ищем
}
```

```
double df(double x)
{
    return 2 - sin(x);
}

// a, b - пределы хорды, epsilon – необходимая погрешность
double chordMethod(double a, double b, double epsilon)
{
    int counter = 1;
    while (abs(b - a) > epsilon) // пока отрезок больше епсилон
    {
        InputTable(a, b, counter);
        a = a - (b - a) * function(a) / (function(b) - function(a));
        b = b - (a - b) * function(b) / (function(a) - function(b));
        counter++;
    }
    // a, b – (i - 1)-й и i-й члены
    cout << "\nКорень через метод хорд = ";
    return b;
}
double HalfDivisionMethod(double LeftPoint, double RightPoint, double epsilon)
{
    int iteration = 1;
    double midPoint = 0.0;
    if (function(LeftPoint) * function(RightPoint) < 0)// проверка на разность знаков
функции на концах отрезка
    {
        while (abs(RightPoint - LeftPoint) > epsilon)// пока интервал больше
погрешности
        {
            midPoint = (RightPoint + LeftPoint) / 2;
            InputTable(LeftPoint, RightPoint, iteration);
            if (function(LeftPoint) * function(midPoint) < 0)  RightPoint = midPoint;//
если функция имеет разные знаки, то правая точка середина отрезка
            else LeftPoint = midPoint;
            //midPoint = (RightPoint + LeftPoint) / 2;
            iteration++;
        }
    }
    else
    {
        cout << "Интервал выбран неверно. Функция имеет одинаковые знаки на концах
отрезка" << endl;
    }
    cout << "\nКорень через метод половинного деления = ";
    return midPoint;
}

void findGraficalSolution(float& left, float& right) // отделяем корни
```

```
{
    for (float x = -1; x < 5; x += 0.01)
    {
        if (ceil(function(x)) == 0)
        {
            left = x - 1.0;
            right = x + 1.0;

        }
    }
}


double NewtownMethod(double x0, double epsilon)
{
    double x;

    for (int i = 1; abs(function(x0)) >= epsilon && i < 10; i++)
    {
        x = x0 - function(x0) / df(x0);
        InputNewtonMethod(x, x0, i);
        x0 = x;
    }
    cout << "\nКорень через метод Ньютона = ";
    return x;
}


int main(void)
{
    setlocale(LC_ALL, "Rus");
    float left = 0;
    float right = 0;
    float x0 = 10;
    float eps = 0.0001;

    findGraficalSolution(left, right);// отделяем корни

    cout << "Метод Ньютона" << endl;
    cout << NewtownMethod(x0, eps) << "\n" << endl;
    cout << "Метод половинного деления\n" << endl;
    cout << HalfDivisionMethod(left, right, eps) << endl;
    cout << "\nМетод хорд\n" << endl;
    cout << chordMethod(left, right, eps) << endl;

    return 0;
}
```

GO

```
package main
```

```
import (
    "fmt"
    "math"
)

func InputTable(leftPoint, rightPoint float64, N int) {
    if N == 1 {
        fmt.Println("N\t     a \t\t\t     b \t\t\t     b-a")
        fmt.Println("----------------------------------------------------------------------
----------------")
    }
    fmt.Printf("%d |\t%.10f\t\t%.10f\t\t%.12f\n", N, leftPoint, rightPoint, rightPoint-
leftPoint)
}

func InputNewtonMethod(leftPoint, rightPoint float64, N int) {
    if N == 1 {
        fmt.Println("N\t     Xn \t\t\t     Xn+1 \t\t\t     Xn+1-Xn")
        fmt.Println("----------------------------------------------------------------------
----------------")
    }
    fmt.Printf("%d |\t%.10f\t\t%.10f\t\t%.12f\n", N, leftPoint, rightPoint, rightPoint-
leftPoint)
}

func function(x float64) float64 {
    return 2*x + math.Cos(x) // Заменить функцией, корни которой мы ищем
}

func df(x float64) float64 {
    return 2 - math.Sin(x)
}

func chordMethod(a, b, epsilon float64) float64 {
    counter := 1
    for math.Abs(b-a) > epsilon { // пока отрезок больше епсилон
        InputTable(a, b, counter)
        a = a - (b-a)*function(a)/(function(b)-function(a))
        b = b - (a-b)*function(b)/(function(a)-function(b))
        counter++
    }
    // a, b – (i - 1)-й и i-й члены
    fmt.Println("\nКорень через метод хорд =", b)
    return b
}

func HalfDivisionMethod(leftPoint, rightPoint, epsilon float64) float64 {
    iteration := 1
    var midPoint float64
```

```
    if function(leftPoint)*function(rightPoint) < 0 { // проверка на разность знаков функции на концах отрезка
        for math.Abs(rightPoint-leftPoint) > epsilon { // пока интервал больше погрешности
            midPoint = (rightPoint + leftPoint) / 2
            InputTable(leftPoint, rightPoint, iteration)
            if function(leftPoint)*function(midPoint) < 0 {
                rightPoint = midPoint // если функция имеет разные знаки, то правая точка середина отрезка
            } else {
                leftPoint = midPoint
            }
            iteration++
        }
    } else {
        fmt.Println("Интервал выбран неверно. Функция имеет одинаковые знаки на концах отрезка")
    }
    fmt.Println("\nКорень через метод половинного деления =", midPoint)
    return midPoint
}

func findGraficalSolution(left, right *float64) {
    for x := -1.0; x < 5; x += 0.01 {
        if math.Ceil(function(x)) == 0 {
            *left = x - 1.0
            *right = x + 1.0
        }
    }
}

func NewtownMethod(x0, epsilon float64) float64 {
    var x float64
    for i := 1; math.Abs(function(x0)) >= epsilon && i < 10; i++ {
        x = x0 - function(x0)/df(x0)
        InputNewtonMethod(x, x0, i)
        x0 = x
    }
    fmt.Println("\nКорень через метод Ньютона =", x)
    return x
}

func main() {
    var left, right float64
    x0 := 10.0
    eps := 0.0001

    findGraficalSolution(&left, &right) // отделяем корни

    fmt.Println("Метод Ньютона")
```

```
    NewtownMethod(x0, eps)
    fmt.Println("\nМетод половинного деления")
    HalfDivisionMethod(left, right, eps)
    fmt.Println("\nМетод хорд")
    chord
```

Результат работы программы:

```
Метод Ньютона
N            Xn                              Xn+1                              Xn+1-Xn
-------------------------------------------------------------------------------------
1 |         2.46825                        10                      7.53175
2 |       -0.550323                   2.46825                      3.01857
3 |       -0.451911                  -0.550323                    -0.0984122
4 |       -0.450184                  -0.451911                   -0.00172662

Корень через метод Ньютона = -0.450184

Метод половинного деления

N            a                          b                          b-a
-------------------------------------------------------------------------------------
1 |           -1.46                  0.539999                             2
2 |       -0.460001                  0.539999                             1
3 |       -0.460001                 0.0399995                           0.5
4 |       -0.460001                 -0.210001                          0.25
5 |       -0.460001                 -0.335001                         0.125
6 |       -0.460001                 -0.397501                        0.0625
7 |       -0.460001                 -0.428751                       0.03125
8 |       -0.460001                 -0.444376                      0.015625
9 |       -0.452188                 -0.444376                     0.0078125
10 |      -0.452188                 -0.448282                    0.00390625
11 |      -0.450235                 -0.448282                    0.00195312
12 |      -0.450235                 -0.449258                   0.000976562
13 |      -0.450235                 -0.449747                   0.000488281
14 |      -0.450235                 -0.449991                   0.000244141
15 |      -0.450235                 -0.450113                    0.00012207

Корень через метод половинного деления = -0.450174

Метод хорд

N            a                          b                          b-a
-------------------------------------------------------------------------------------
1 |           -1.46                  0.539999                             2
2 |       -0.276369                  -0.494998                     -0.218629
3 |        -0.44868                  -0.450171                    -0.00149157

Корень через метод хорд = -0.450184
```

Задание 2:

C++

```
#include <iostream>
#include <random>
#include <vector>
```

```cpp
#include <algorithm>

using namespace std;

int GetRandomNumber(int min, int max)
{
    random_device rd;//random_device, который является источником недетерминированных случайных чисел.
    //Затем мы используем это устройство для заполнения генератора std::minstd_rand. Затем функция генератора() используется для генерации случайного числа
    minstd_rand generator(rd());

    uniform_int_distribution<int> distribution(min, max);// функция destribition для задания диапозона значений
    int random_number = distribution(generator);
    return random_number;
}

void task2_2and2_3(vector <int>& arr1, vector <int>& arr2, int sizeArr)
{
    cout << "Task 2_2 and 2_3\n\n";

    for (auto& i : arr1)
    {
        i = GetRandomNumber(0, 100);
        cout << i << " ";
    }

    int max_element = 0, index_max = 0, sum=0;
    int min_element = 100, index_min = 0;
    int counterForArr2 = 0;

    for (int i = 0; i < arr1.size(); i++)
    {
        if (arr1[i] > max_element)
        {
            max_element = arr1[i];
            index_max = i;
        }
        if (arr1[i] < min_element)
        {
            min_element = arr1[i];
            index_min = i;
        }
    }
    cout << "\nMax element = " << max_element<<"| " << index_max << "\nMin Element = " << min_element <<"| " << index_min<< endl;
    if (index_max < index_min)
    {
        for (int i = index_max+1; i <index_min; i++, counterForArr2++)
```

```
        {
            sum += arr1[i];
            arr2[counterForArr2] = arr1[i];
        }
    }
    else
    {
        for (int i = index_min + 1; i < index_max; i++, counterForArr2++)
        {
            sum += arr1[i];
            arr2[counterForArr2] = arr1[i];
        }

    }
    cout << "Сумма элементов равна " << sum << endl;
    for (; counterForArr2 < sizeArr; counterForArr2++)
    {
        arr2[counterForArr2] = arr1[GetRandomNumber(0, sizeArr - 1)];
    }
    for (auto i : arr2)
    {
        cout << i << " ";
    }
    cout << endl;
}

void task2_4(int size)// сортировка элементов на четных места по возрастанию
{
    cout << "\nTask 2_4\n\n";
    vector<char> arrChar(size);
    for (auto& i : arrChar)
    {
        int sumvol = GetRandomNumber(65, 122);
        i = static_cast<char>(sumvol);
        cout << i << " ";
    }
    cout << endl;
    for (size_t i = 0; i < arrChar.size(); i += 2) {
        size_t minIndex = i;
        for (size_t j = i + 2; j < arrChar.size(); j += 2) {
            if (arrChar[j] < arrChar[minIndex]) {
                minIndex = j;
            }
        }
        std::swap(arrChar[i], arrChar[minIndex]);
    }
    for (auto item : arrChar)
    {
        cout << item << " ";
    }
```

```
}

void task2_5(int size)
{
    cout << "\n\nTask 2_5\n\n";

    vector<int> Array(size);
    vector<int> NewArray(size);

    for (auto& i : Array)
    {
        i = GetRandomNumber(100, 900);
        cout << i << " ";
    }

    cout << endl;

    sort(Array.begin(), Array.end());

    cout << "\nОтсортированный массив" << endl;

    for (auto& i : Array)// вывод отсортированного массива
    {
        cout << i << " ";
    }
    cout <<"\n" << endl;

    for (auto& item : NewArray)//заполняем новый массив случайными элементами из
первого
    {
        item = Array[GetRandomNumber(0, size - 1)];
    }
    for (int i = 0; i < size; i++)
    {
        if (NewArray[i] < Array[i]) NewArray[i] = 0;//если в новом массив элемент
меньше исходного, то записываем единицу
    }
    for (auto item : NewArray)
    {
        cout << item << " ";
    }
}

int main(void)
{
```

```
    setlocale(LC_ALL, "rus");
    int sizeArr1 = 20;

    vector <int> arr1(sizeArr1);
    vector <int> arr2(sizeArr1);

    task2_2and2_3(arr1, arr2,sizeArr1);
    task2_4(sizeArr1);
    task2_5(sizeArr1);

    return 0;

}
```

GO

```
package main

import (
    "fmt"
    "math/rand"
    "time"
)

func GetRandomNumber(min, max int) int {
    rand.Seed(time.Now().UnixNano())
    return rand.Intn(max-min+1) + min
}

func task2_2and2_3(arr1, arr2 []int, sizeArr int) {
    fmt.Println("Task 2_2 and 2_3\n")

    for i := range arr1 {
        arr1[i] = GetRandomNumber(0, 100)
        fmt.Print(arr1[i], " ")
    }

    maxElement, indexMax := 0, 0
    minElement, indexMin := 100, 0
    sum := 0
    counterForArr2 := 0

    for i := range arr1 {
        if arr1[i] > maxElement {
            maxElement = arr1[i]
            indexMax = i
        }
        if arr1[i] < minElement {
            minElement = arr1[i]
            indexMin = i
        }
```

```
    }
    fmt.Printf("\nMax element = %d| %d\nMin Element = %d| %d\n", maxElement, indexMax,
minElement, indexMin)

    if indexMax < indexMin {
        for i := indexMax + 1; i < indexMin; i++ {
            sum += arr1[i]
            arr2[counterForArr2] = arr1[i]
            counterForArr2++
        }
    } else {
        for i := indexMin + 1; i < indexMax; i++ {
            sum += arr1[i]
            arr2[counterForArr2] = arr1[i]
            counterForArr2++
        }
    }

    fmt.Println("Сумма элементов равна", sum)

    for ; counterForArr2 < sizeArr; counterForArr2++ {
        arr2[counterForArr2] = arr1[GetRandomNumber(0, sizeArr-1)]
    }

    for _, i := range arr2 {
        fmt.Print(i, " ")
    }
    fmt.Println()
}

func task2_4(size int) {
    fmt.Println("\nTask 2_4\n")
    arrChar := make([]rune, size)
    for i := range arrChar {
        arrChar[i] = rune(GetRandomNumber(65, 122))
        fmt.Print(string(arrChar[i]), " ")
    }
    fmt.Println()

    for i := 0; i < len(arrChar); i += 2 {
        minIndex := i
        for j := i + 2; j < len(arrChar); j += 2 {
            if arrChar[j] < arrChar[minIndex] {
                minIndex = j
            }
        }
        arrChar[i], arrChar[minIndex] = arrChar[minIndex], arrChar[i]
    }

    for _, item := range arrChar {
```

```
        fmt.Print(string(item), " ")
    }
}

func task2_5(size int) {
    fmt.Println("\n\nTask 2_5\n")

    Array := make([]int, size)
    NewArray := make([]int, size)

    for i := range Array {
        Array[i] = GetRandomNumber(100, 900)
        fmt.Print(Array[i], " ")
    }

    fmt.Println("\nОтсортированный массив")
    sort(Array)

    for _, i := range Array {
        fmt.Print(i, " ")
    }
    fmt.Println()

    for i := range NewArray {
        NewArray[i] = Array[GetRandomNumber(0, size-1)]
    }

    for i := range NewArray {
        if NewArray[i] < Array[i] {
            NewArray[i] = 0
        }
    }

    for _, item := range NewArray {
        fmt.Print(item, " ")
    }
}

func sort(arr []int) {
    for i := 0; i < len(arr); i++ {
        for j := i + 1; j < len(arr); j++ {
            if arr[i] > arr[j] {
                arr[i], arr[j] = arr[j], arr[i]
            }
        }
    }
}

func main() {
    sizeArr := 20
```

```
    arr1 := make([]int, sizeArr)
    arr2 := make([]int, sizeArr)

    task2_2and2_3(arr1, arr2, sizeArr)
    task2_4(sizeArr)
    task2_5(sizeArr)
}
```

Результат работы:

```
Task 2_2 and 2_3

96 0 32 8 24 16 16 69 52 15 68 3 88 2 5 1 66 91 98 35
Max element = 98| 18
Min Element = 0| 1
Сумма элементов равна 556
32 8 24 16 16 69 52 15 68 3 88 2 5 1 66 91 3 96 88 91

Task 2_4

I ] t v b y N w M i n Y a d P v S u y y
I ] M v N y P w S i a Y b d n v t u y y

Task 2_5

832 754 843 675 721 616 143 423 146 631 818 108 237 850 642 720 774 113 173 308

Отсортированный массив
108 113 143 146 173 237 308 423 616 631 642 675 720 721 754 774 818 832 843 850

850 843 675 0 774 818 818 774 0 0 850 721 0 721 0 843 0 0 0 0
```

Задание 3:

C++

```
#include <iostream>
#include <vector>
#include <random>
#include <cmath>
#include <locale.h>
using namespace std;

void fillingTheVector(vector<int>& vec) //заполнение вектора случайными числами
{
    random_device rd;//random_device, который является источником недетерминированных случайных чисел.
    //Затем мы используем это устройство для заполнения генератора std::minstd_rand. Затем функция генератора() используется для генерации случайного числа
    minstd_rand generator(rd());
```

```
    uniform_int_distribution<int> distribution(0, 100);// функция destribition для
задания диапозона значений

    for (int& item : vec)
    {
        item = distribution(generator);
    }
}

int ElementCountInVector(const vector<int>& container, int element) //подсчет
количества вхождений заданного элемента в вектор
{
    int count = 0;
    for (const int& value : container)
    {
        if (value == element) count++;
    }
    return count;
}

double mathExpectation(vector<int> container) //вычисление реального математического
ожидания
{
    float Mx = 0.0;
    for (auto i = 0; i < container.size(); i++)
    {
        Mx += container[i] * 0.01;// диапозон чисел 100. Каждое равновероятно
    }
    return Mx;
}

double mathExp(vector <int> container) //вычисление ожидаемого математического ожидания
{
    float Mx0 = 0.0;
    Mx0 = container.size()/2;//  при равномерном распределении случайной величины в
диапазоне от a до b, математическое ожидание вычисляется как среднее арифметическое
этих границ
    return Mx0;
}

double mathDispersion(vector<int> container) //функция для вычисления дисперсии
{
    float Mx = mathExpectation(container);// вычисляем дисперсию относительно реального
мат ожидания
    float sum = 0.0;
    for (int i = 0; i < container.size(); i++)
    {
        sum += pow(container[i] - Mx, 2);// суммируем разницу между всеми значениями и
мат ожиданием в квадрате
    }
```

```
    return sum / (container.size()-1);//делим на количество в выборке- 1
}

double tabLaplas(double x) //функции Лапласа (стандартного нормального распределения) для заданного аргумента x
{
    return (1.0 / 2.0) * (1.0 + erf(x / sqrt(2.0)));
}

double chiSquare(const vector<int>& container)
{
    double Mx = mathExpectation(container);
    double chi = 0.0;
    for (int i = 0; i < 10; i++) // Перебор 10 интервалов
    {
        int observed = ElementCountInVector(container, i * 10); // Наблюдаемая частота
        double expected = container.size() * 0.1; // Ожидаемая частота

        chi += pow(observed - expected, 2) / expected;
    }
    return chi;
}
double quantile_chi_square(double p, int df)
{
    // Функция для вычисления квантили распределения хи-квадрат
    // Реализация на основе библиотеки math.h
    return sqrt(2.0 * tgamma(df / 2.0) * pow(p, 1.0 / df) / tgamma((df - 1.0) / 2.0));
}
double chiSquareCriticalValue(int df, double alpha)
{
    // Функция для вычисления критического значения хи-квадрат
    double p = 1.0 - alpha; // Вероятность, соответствующая уровню значимости
    return quantile_chi_square(p, df); // Вычисление критического значения
}



int main()
{
    setlocale(LC_ALL, "Rus");

    vector<int> vector50(50);
    fillingTheVector(vector50);

    vector<int> vector100(100);
    fillingTheVector(vector100);

    vector<int> vector1000(1000);
    fillingTheVector(vector1000);
```

```
    double criticalValue = chiSquareCriticalValue(7, 0.05); // Критическое значение хи-квадрат для 99 степеней свободы и уровня значимости 0.05

   // для 50 элементов
    double chiSquareValue = chiSquare(vector50);
    double meanExpected = mathExpectation(vector50); //реальное математическое ожидание
    double meanObserved = mathExp(vector50);// ожидаемое мат ожидание

    cout << "X^2: " << chiSquareValue << endl;

    if (chiSquareValue <= criticalValue)
    {
        cout << "Гипотеза о нормальном распределении не отвергается." << endl;
    }
    else
    {
        cout << "Гипотеза о нормальном распределении отвергается." << endl;
    }

    cout << "Ожидаемое математическое ожидание: " << meanObserved << endl;
    cout << "Реальное математическое ожидание: " << meanExpected << endl;
    cout << endl;

    //для 100 элементов
    chiSquareValue = chiSquare(vector100);
    meanExpected = mathExpectation(vector100);
    meanObserved = mathExp(vector100);

    cout << "X^2: " << chiSquareValue << endl;

    if (chiSquareValue <= criticalValue)
    {
        cout << "Гипотеза о нормальном распределении не отвергается." << endl;
    }
    else
    {
        cout << "Гипотеза о нормальном распределении отвергается." << endl;
    }

    cout << "Ожидаемое математическое ожидание: " << meanObserved << endl;
    cout << "Реальное математическое ожидание: " << meanExpected << endl;
    cout << endl;

    //для 1000 элементов
    chiSquareValue = chiSquare(vector1000);
    meanExpected = mathExpectation(vector1000);
    meanObserved = mathExp(vector1000);

    cout << "X^2: " << chiSquareValue << endl;
```

```
    if (chiSquareValue <= criticalValue)
    {
        cout << "Гипотеза о нормальном распределении не отвергается." << endl;
    }
    else
    {
        cout << "Гипотеза о нормальном распределении отвергается." << endl;
    }

    cout << "Ожидаемое математическое ожидание: " << meanObserved << endl;
    cout << "Реальное математическое ожидание: " << meanExpected << endl;
    cout << endl;

    return 0;
}
```

GO

```
package main

import (
    "fmt"
    "math"
    "math/rand"
    "time"
)

func fillingTheVector(vec []int) { // заполнение вектора случайными числами
    rand.Seed(time.Now().UnixNano())
    for i := range vec {
        vec[i] = rand.Intn(101)
    }
}

func elementCountInVector(container []int, element int) int { // подсчет количества
вхождений заданного элемента в вектор
    count := 0
    for _, value := range container {
        if value == element {
            count++
        }
    }
    return count
}

func mathExpectation(container []int) float64 { // вычисление реального математического
ожидания
    var Mx float64
    for _, i := range container {
        Mx += float64(i) * 0.01 // диапозон чисел 100. Каждое равновероятно
    }
    return Mx
```

```
}

func mathExp(container []int) float64 { // вычисление ожидаемого математического
ожидания
    return float64(len(container)) / 2 // при равномерном распределении случайной
величины в диапазоне от a до b, математическое ожидание вычисляется как среднее
арифметическое этих границ
}

func mathDispersion(container []int) float64 { // функция для вычисления дисперсии
    Mx := mathExpectation(container) // вычисляем дисперсию относительно реального мат
ожидания
    var sum float64
    for _, i := range container {
        sum += math.Pow(float64(i)-Mx, 2) // суммируем разницу между всеми значениями и
мат ожиданием в квадрате
    }
    return sum / float64(len(container)-1) // делим на количество в выборке- 1
}

func tabLaplas(x float64) float64 { // функции Лапласа (стандартного нормального
распределения) для заданного аргумента x
    return 0.5 * (1 + math.Erf(x/math.Sqrt(2)))
}

func chiSquare(container []int) float64 {

    var chi float64
    for i := 0; i < 10; i++ { // Перебор 10 интервалов
        observed := elementCountInVector(container, i*10) // Наблюдаемая частота
        expected := float64(len(container)) * 0.1         // Ожидаемая частота

        chi += math.Pow(float64(observed)-expected, 2) / expected
    }
    return chi
}

func quantileChiSquare(p float64, df int) float64 {
    // Функция для вычисления квантили распределения хи-квадрат
    // Реализация на основе math.Gamma и math.Pow
    return math.Sqrt(2.0 * math.Gamma(float64(df)/2.0) * math.Pow(p, 1.0/float64(df)) /
math.Gamma((float64(df)-1.0)/2.0))
}

func chiSquareCriticalValue(df int, alpha float64) float64 {
    // Функция для вычисления критического значения хи-квадрат
    p := 1.0 - alpha                // Вероятность, соответствующая уровню значимости
    return quantileChiSquare(p, df) // Вычисление критического значения
}
```

```
func main() {
    rand.Seed(time.Now().UnixNano())

    vector50 := make([]int, 50)
    fillingTheVector(vector50)

    vector100 := make([]int, 100)
    fillingTheVector(vector100)

    vector1000 := make([]int, 1000)
    fillingTheVector(vector1000)

    criticalValue := chiSquareCriticalValue(7, 0.05) // Критическое значение хи-квадрат
для 99 степеней свободы и уровня значимости 0.05

    // для 50 элементов
    chiSquareValue := chiSquare(vector50)
    meanExpected := mathExpectation(vector50) // реальное математическое ожидание
    meanObserved := mathExp(vector50)         // ожидаемое мат ожидание

    fmt.Println("X^2:", chiSquareValue)

    if chiSquareValue <= criticalValue {
        fmt.Println("Гипотеза о нормальном распределении не отвергается.")
    } else {
        fmt.Println("Гипотеза о нормальном распределении отвергается.")
    }

    fmt.Println("Ожидаемое математическое ожидание:", meanObserved)
    fmt.Println("Реальное математическое ожидание:", meanExpected)
    fmt.Println()

    // для 100 элементов
    chiSquareValue = chiSquare(vector100)
    meanExpected = mathExpectation(vector100)
    meanObserved = mathExp(vector100)

    fmt.Println("X^2:", chiSquareValue)

    if chiSquareValue <= criticalValue {
        fmt.Println("Гипотеза о нормальном распределении не отвергается.")
    } else {
        fmt.Println("Гипотеза о нормальном распределении отвергается.")
    }

    fmt.Println("Ожидаемое математическое ожидание:", meanObserved)
    fmt.Println("Реальное математическое ожидание:", meanExpected)
    fmt.Println()

    // для 1000 элементов
```

```
    chiSquareValue = chiSquare(vector1000)
    meanExpected = mathExpectation(vector1000)
    meanObserved = mathExp(vector1000)

    fmt.Println("X^2:", chiSquareValue)

    if chiSquareValue <= criticalValue {
        fmt.Println("Гипотеза о нормальном распределении не отвергается.")
    } else {
        fmt.Println("Гипотеза о нормальном распределении отвергается.")
    }

    fmt.Println("Ожидаемое математическое ожидание:", meanObserved)
    fmt.Println("Реальное математическое ожидание:", meanExpected)
    fmt.Println()
}
```

Результат работы программы:

```
X^2: 35.4
Гипотеза о нормальном распределении отвергается.
Ожидаемое математическое ожидание: 25
Реальное математическое ожидание: 25.28

X^2: 88.6
Гипотеза о нормальном распределении отвергается.
Ожидаемое математическое ожидание: 50
Реальное математическое ожидание: 54.36

X^2: 805.37
Гипотеза о нормальном распределении отвергается.
Ожидаемое математическое ожидание: 500
Реальное математическое ожидание: 490.11
```

Задание 4:

C++

```
#include <iostream>
#include <vector>
#include <random>
#include <locale.h>
using namespace std;

// Алгоритм "Доверчивый"
bool confiding(int round_number, vector<bool> self_choices, vector<bool> enemy_choices)
{
    // Всегда сотрудничает, независимо от предыдущих ходов
    return true;
}

// Алгоритм "Мстительный"
```

```
bool revengeful(int round_number, vector<bool> self_choices, vector<bool>
enemy_choices)
{
    // Сотрудничает, если в предыдущем раунде противник также сотрудничал, иначе
предает
    if (round_number == 0 || enemy_choices[round_number - 1])
    {
        return true;
    }
    else {
        return false;
    }
}

// Алгоритм "Осторожный"
bool careful(int round_number, vector<bool> self_choices, vector<bool> enemy_choices)
{
    // Сотрудничает, если в предыдущих 3 раундах противник 2 раза или больше
сотрудничал, иначе предает
    int cooperationCount = 0;
    for (int i = max(0, round_number - 3); i < round_number; i++)
    {
        if (enemy_choices[i])
        {
            cooperationCount++;
        }
    }
    return cooperationCount >= 2;
}

int main()
{
    setlocale(LC_ALL, "Rus");
    // Генерируем случайное количество раундов от 100 до 200
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> distribution(100, 200);
    int total_rounds = distribution(gen);

    // Массивы для хранения выборов игроков
    vector<bool> player1_choices(total_rounds, false);
    vector<bool> player2_choices(total_rounds, false);

    // Выбор алгоритмов для игроков
    bool (*player1_algorithm)(int, vector<bool>, vector<bool>) = &revengeful;
    bool (*player2_algorithm)(int, vector<bool>, vector<bool>) = &confiding;

    // Игровой цикл
    int player1_score = 0, player2_score = 0;
    for (int round = 0; round < total_rounds; round++) {
```

```
        player1_choices[round] = (*player1_algorithm)(round, player1_choices,
player2_choices);
        player2_choices[round] = (*player2_algorithm)(round, player2_choices,
player1_choices);

        if (player1_choices[round] && player2_choices[round])
        {
            player1_score += 24;
            player2_score += 24;
        }
        else if (player1_choices[round] && !player2_choices[round])
        {
            player2_score += 20;
        }
        else if (!player1_choices[round] && player2_choices[round])
        {
            player1_score += 20;
        }
        else
        {
            player1_score += 4;
            player2_score += 4;
        }
    }

    // Вывод результатов
    cout << "Количество раундов: " << total_rounds << endl;
    cout << "Счет игрока 1: " << player1_score << endl;
    cout << "Счет игрока 2: " << player2_score << endl;

    return 0;
}
```

GO

```
package main

import (
    "fmt"
    "math/rand"
    "time"
)

// Алгоритм "Доверчивый"
func confiding(roundNumber int, selfChoices, enemyChoices []bool) bool {
    // Всегда сотрудничает, независимо от предыдущих ходов
    return true
}

// Алгоритм "Мстительный"
func revengeful(roundNumber int, selfChoices, enemyChoices []bool) bool {
```

```
    // Сотрудничает, если в предыдущем раунде противник также сотрудничал, иначе
предает
    if roundNumber == 0 || enemyChoices[roundNumber-1] {
        return true
    }
    return false
}

// Алгоритм "Осторожный"
func careful(roundNumber int, selfChoices, enemyChoices []bool) bool {
    // Сотрудничает, если в предыдущих 3 раундах противник 2 раза или больше
сотрудничал, иначе предает
    cooperationCount := 0
    for i := max(0, roundNumber-3); i < roundNumber; i++ {
        if enemyChoices[i] {
            cooperationCount++
        }
    }
    return cooperationCount >= 2
}

func max(a, b int) int {
    if a > b {
        return a
    }
    return b
}

func main() {
    // Генерируем случайное количество раундов от 100 до 200
    rand.Seed(time.Now().UnixNano())
    totalRounds := rand.Intn(101) + 100

    // Массивы для хранения выборов игроков
    player1Choices := make([]bool, totalRounds)
    player2Choices := make([]bool, totalRounds)

    // Выбор алгоритмов для игроков
    player1Algorithm := revengeful
    player2Algorithm := confiding

    // Игровой цикл
    player1Score, player2Score := 0, 0
    for round := 0; round < totalRounds; round++ {
        player1Choices[round] = player1Algorithm(round, player1Choices, player2Choices)
        player2Choices[round] = player2Algorithm(round, player2Choices, player1Choices)

        if player1Choices[round] && player2Choices[round] {
            player1Score += 24
            player2Score += 24
```

```
        } else if player1Choices[round] && !player2Choices[round] {
            player2Score += 20
        } else if !player1Choices[round] && player2Choices[round] {
            player1Score += 20
        } else {
            player1Score += 4
            player2Score += 4
        }
    }

    // Вывод результатов
    fmt.Println("Количество раундов:", totalRounds)
    fmt.Println("Счет игрока 1:", player1Score)
    fmt.Println("Счет игрока 2:", player2Score)
}
```

Результат работы программы:

```
X^2: 35.4
Гипотеза о нормальном распределении отвергается.
Ожидаемое математическое ожидание: 25
Реальное математическое ожидание: 25.28

X^2: 88.6
Гипотеза о нормальном распределении отвергается.
Ожидаемое математическое ожидание: 50
Реальное математическое ожидание: 54.36

X^2: 805.37
Гипотеза о нормальном распределении отвергается.
Ожидаемое математическое ожидание: 500
Реальное математическое ожидание: 490.11
```

Задание 5:

C++

```
#include <iostream>
#include <vector>
#include <locale.h>

using namespace std;

int main()
{
    setlocale(LC_ALL, "Rus");
    int A = 0, B = 0, C = 0, X0 = 0, Xi = 0;;
    vector<int> arr1;
    cout << "Введите диапозон значений C:\nC = ";
    cin >> C;
    cout << "Введите множитель A(0<=A<=C):\nA = ";
    cin >> A;
    cout << "Введите инкрементирующее значений B(0<=B<=C):\nB = ";
```

```
    cin >> B;
    cout << "Введите начальное значений X0(0<=X0<=C):\nX0 = ";
    cin >> X0;
    arr1.emplace(arr1.begin(), X0);
    for (int i = 1; i < C; i++)
    {
        Xi = (A * arr1[i - 1] + B) % C;
        arr1.emplace(arr1.begin() + i, Xi);
    }
    for (auto i : arr1)
    {
        cout << i << " ";
    }
    for (size_t i = 1; i<arr1.size(); i++)
    {
        if (arr1[0] == arr1[i])
        {
            cout << "\nИндекс начала повторяющейся последовательности = " << i+1 <<
endl;
            break;
        }
    }

    return 0;
}
```

GO

```
package main

import (
    "fmt"
)

func main() {
    var A, B, C, X0, Xi int
    var arr1 []int

    fmt.Println("Введите диапозон значений C:")
    fmt.Print("C = ")
    fmt.Scan(&C)

    fmt.Println("Введите множитель A(0<=A<=C):")
    fmt.Print("A = ")
    fmt.Scan(&A)

    fmt.Println("Введите инкрементируещее значений B(0<=B<=C):")
    fmt.Print("B = ")
    fmt.Scan(&B)

    fmt.Println("Введите начальное значений X0(0<=X0<=C):")
    fmt.Print("X0 = ")
```

```
    fmt.Scan(&X0)

    arr1 = append(arr1, X0)
    for i := 1; i < C; i++ {
        Xi = (A*arr1[i-1] + B) % C
        arr1 = append(arr1, Xi)
    }

    for _, i := range arr1 {
        fmt.Print(i, " ")
    }
    fmt.Println()

    for i := 1; i < len(arr1); i++ {
        if arr1[0] == arr1[i] {
            fmt.Println("Индекс начала повторяющейся последовательности =", i+1)
            break
        }
    }
}
```

Результат работы программы:

```
Введите диапозон значений C:
C = 17
Введите множитель A(0<=A<=C):
A = 15
Введите инкрементирующее значений B(0<=B<=C):
B = 14
Введите начальное значений X0(0<=X0<=C):
X0 = 2
2 10 11 9 13 5 4 6 2 10 11 9 13 5 4 6 2
Индекс начала повторяющейся последовательности = 9
```