МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра защиты информации

**

**Лабораторная работа №3**

«Изучение циклических алгоритмов, операторов цикла,

программирование циклического вычислительного процесса»

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

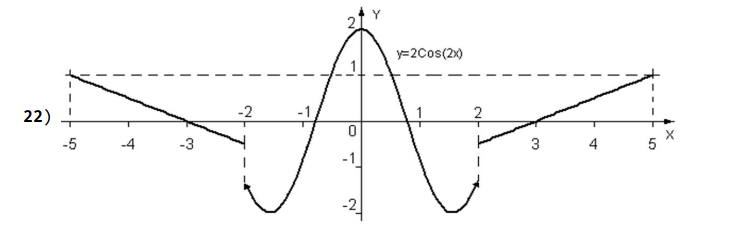
|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБ-323», «АВТФ»  *Раков Михаил Андреевич*  «28» апреля 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Исаев Глеб Андреевич*  «29» апреля 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

**Цели и задачи работы**: изучение циклических алгоритмов, операторов цикла, программирование циклического вычислительного процесса (ЯП С++ и другой (на выбор, из числа ЛР2)).

**Задание к работе**: реализовать циклический вычислительный процесс. Самостоятельно решить задачи в соответствии с индивидуальным вариантом.

**Задание № 1 (вариант 22)**

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. Таблица должна иметь заголовком и шапку.  
Параметры, необходимые для решения задания следует получить из графика и определить в программе.

**Код программы (на С++)**

#include <iostream>

#include <iomanip>

#include <sstream>

#include <string>

#include <fstream>

#include <cmath>

using namespace std;

double function (double x);

size\_t columns\_width (size\_t titleWidth, double x0, double x1, int prec)

{

size\_t x0\_len = (stringstream {} << fixed << setprecision(prec) << x0).str().size();

size\_t x1\_len = (stringstream {} << fixed << setprecision(prec) << x1).str().size();

return max(titleWidth, max(x0\_len, x1\_len));

}

int main()

{

double x0 = 3.12345, x1 = 6, step = 0.1;

string title\_arg = "x:";

string title\_res = "results:";

size\_t width = columns\_width(title\_arg.size(), x0, x1, 3) + 1;

cout << "Table" << endl;

cout << left << setw(width) << title\_arg << title\_res << endl;

cout << fixed << setprecision(3);

for (; x0 <= x1; x0 += step)

{

x0 = (abs(x0) < step / 10) ? 0 : x0;

cout << left << setw(width) << x0 << function(x0) << endl;

}

}

double func1 (double x)

{

return -(x + 3) / 2;

}

double func2 (double x)

{

return 2 \* cos(2 \* x);

}

double func3 (double x)

{

return (x + 3) / 2;

}

double function (double x)

{

if (x <= -2)

{

return func1 (x);

}

else if (-2 < x and x < 2)

{

return func2 (x);

}

else

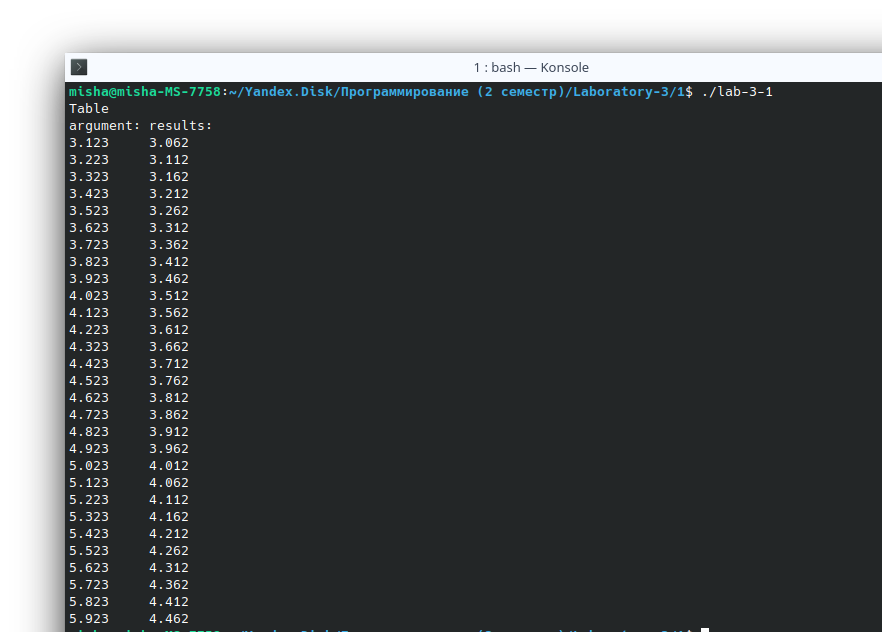
{

return func3 (x);

}

}

**Результат работы программы**



**Код программы (на Haskell)**

import Text.Printf

import Data.List

function :: Double -> Double

function x

| x <= -2 = func1 x

| -2 < x && x < 2 = func2 x

| otherwise = func3 x

columnsWidth :: Int -> Double -> Double -> Int -> Int

columnsWidth titleWidth x0 x1 prec = maximum [titleWidth, x0Len, x1Len]

where

x0Len = length $ printf "%.\*f" prec x0

x1Len = length $ printf "%.\*f" prec x1

main :: IO ()

main = do

let x0 = 3.12345

x1 = 6

step = 0.1

titleArg = "x:"

titleRes = "results:"

width = columnsWidth (length titleArg) x0 x1 3 + 1

putStrLn "Table"

putStrLn $ printf "%-\*s%s" width titleArg titleRes

forM\_ [x0, x0 + step .. x1] $ \x -> do

let newX = if abs x < step / 10 then 0 else x

putStrLn $ printf "%-\*.\*f%.\*f" width 3 newX (function newX)

func1 :: Double -> Double

func1 x = -(x + 3) / 2

func2 :: Double -> Double

func2 x = 2 \* cos (2 \* x)

func3 :: Double -> Double

func3 x = (x + 3) / 2

**Задание № 2**

Тесты на простоту (реализовать метод перебора делителей, тесты Миллера, Поклингтона, ГОСТ Р 34.10-94.

1) Построить таблицу простых чисел, меньших 500, с помощью решета Эратосфена. С использованием этой таблицы:

а) реализовать процедуру получения простых чисел заданной длины на основе теста Миллера;

б) реализовать процедуру получения простых чисел заданной длины на основе теста Поклингтона;

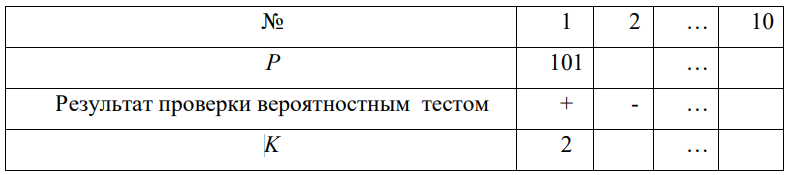
в) реализовать процедуру генерации простых чисел заданной длины ГОСТ Р 34.10-94.

2) Построить 10 простых чисел с помощью полученной процедуры.

3) Каждое построенное число проверить на простоту вероятностным тестом, реализованным в задании к разделу 2. Количество итераций вероятностного теста должно быть таково, чтобы вероятность ошибки не превышала 0,1.

4) Каждое отвергнутое тестом из пункта 1 число проверить вероятностным тестом. Подсчитать k – количество отвергнутых чисел, определенных вероятностным тестом как простые.

5) Результат оформить в виде таблицы:



Здесь № - номер эксперимента, p – построенное простое число, в третьей строке результат проверки построенного числа вероятностным тестом (+ или -), k – количество отвергнутых чисел, определенных вероятностным тестом как простые.

Количество итераций вероятностного теста должно быть таково, чтобы вероятность ошибки не превышала 0,1.

**Код программы (Haskell)**

#include <iostream>

using namespace std;

int main()

{

int nextRound, countMatch = 0;

cin >> nextRound;

while (nextRound > 1)

{

if (nextRound % 2)

{

nextRound = (nextRound - 1) / 2 + 1;

countMatch += (nextRound - 1);

} else {

nextRound /= 2;

countMatch += nextRound;

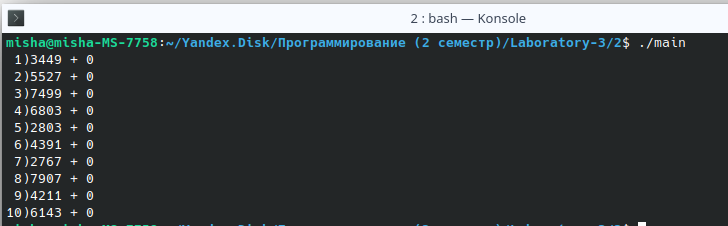
}

}

cout << countMatch;

}

**Результаты работы программы**



**Задание № 3**

Задача на остывание кофе, получить таблицу время/температуру, аппроксимирующую прямую и коэффициент корреляции.

**Код программы (на С++)**

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

using coffee\_type = vector<pair<double, double>>;

coffee\_type coffee (double Tsr, double Tc, double coefHeat, int t)

{

coffee\_type coffeeInTime;

for (int i = 0; i < t; ++i)

{

coffeeInTime.emplace\_back(i, Tc);

Tc -= (coefHeat \* (Tc - Tsr));

}

return coffeeInTime;

}

pair<double, double> aprox (const coffee\_type& coffeeInTime)

{

double sumXY = 0;

double sumX = 0;

double sumY = 0;

double sumSqrX = 0;

size\_t n = coffeeInTime.size();

for (auto [x, y] : coffeeInTime)

{

sumXY += (x \* y);

sumX += x;

sumY += y;

sumSqrX += pow(x, 2);

}

double a = (n \* sumXY - sumX \* sumY) / (n \* sumSqrX - pow(sumX, 2));

double b = (sumY - a \* sumX) / n;

return make\_pair(a, b);

}

double korrel (const coffee\_type& coffeeInTime)

{

double mediumX = 0;

double mediumY = 0;

for (auto [x, y] : coffeeInTime)

{

mediumX += x;

mediumY += y;

}

mediumX /= coffeeInTime.size();

mediumY /= coffeeInTime.size();

double sumDxDy = 0;

double sumSqrDx = 0;

double sumSqrDy = 0;

for (auto [x, y] : coffeeInTime)

{

sumDxDy += ((x - mediumX) \* (y - mediumY));

sumSqrDx += pow(x - mediumX, 2);

sumSqrDy += pow(y - mediumY, 2);

}

return sumDxDy / sqrt(sumSqrDx \* sumSqrDy);

}

int main()

{

int t;

double Tc, Tsr, coefHeat;

cin >> Tsr >> Tc >> coefHeat >> t;

auto coffeInTime = coffee(Tsr, Tc, coefHeat, t);

auto [a, b] = aprox(coffeInTime);

double k = korrel(coffeInTime);

cout << "Время" << "\t" << "Температура кофе" << endl;

for (auto [time, temperature] : coffeInTime)

{

cout << time << "\t" << temperature << endl;

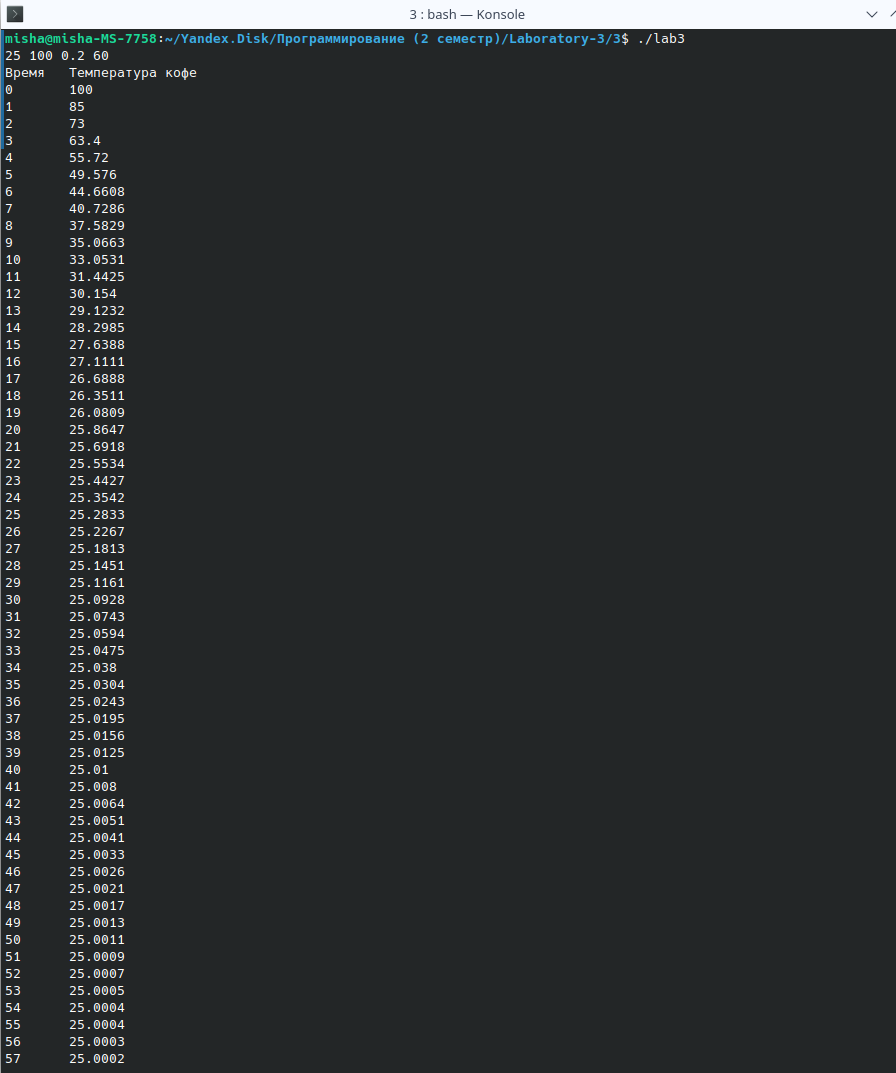
}

cout << "\nЛиния апроксимации: " << "T = " << a << " \* t + " << b << endl;

cout << "\nКоэффициент корреляции " << k << endl;

}

**Результаты работы программы**



**Код программы (на Haskell)**

coffee :: Double -> Double -> Double -> Int -> [(Double, Double)]

coffee Tsr Tc coefHeat t = take t $ iterate (\(i, temp) -> (i+1, temp - coefHeat \* (temp - Tsr))) (0, Tc)

aprox :: [(Double, Double)] -> (Double, Double)

aprox coffeeInTime = (a, b)

where

n = fromIntegral $ length coffeeInTime

sumXY = sum [x \* y | (x, y) <- coffeeInTime]

sumX = sum [x | (x, \_) <- coffeeInTime]

sumY = sum [y | (\_, y) <- coffeeInTime]

sumSqrX = sum [x^2 | (x, \_) <- coffeeInTime]

a = (n \* sumXY - sumX \* sumY) / (n \* sumSqrX - sumX^2)

b = (sumY - a \* sumX) / n

korrel :: [(Double, Double)] -> Double

korrel coffeeInTime = sumDxDy / sqrt (sumSqrDx \* sumSqrDy)

where

n = fromIntegral $ length coffeeInTime

(mediumX, mediumY) = foldl (\(mx, my) (x, y) -> (mx + x, my + y)) (0, 0) coffeeInTime

(sumDxDy, sumSqrDx, sumSqrDy) = foldl (\(sxy, sx, sy) (x, y) -> (sxy + (x - mediumX) \* (y - mediumY), sx + (x - mediumX)^2, sy + (y - mediumY)^2)) (0, 0, 0) coffeeInTime

main :: IO ()

main = do

putStrLn "Enter Tsr, Tc, coefHeat, and t:"

[Tsr, Tc, coefHeat, t] <- fmap (map read . words) getLine

let coffeeInTime = coffee Tsr Tc coefHeat t

(a, b) = aprox coffeeInTime

k = korrel coffeeInTime

putStrLn "Time\tTemperature"

mapM\_ (\(time, temperature) -> putStrLn $ show time ++ "\t" ++ show temperature) coffeeInTime

putStrLn $ "\nLine of approximation: T = " ++ show a ++ " \* t + " ++ show b

putStrLn $ "\nCorrelation coefficient: " ++ show k

**Вывод**

В соответствии с заданием по индивидуальному варианту был реализован циклический вычислительный процесс. Все файлы для лабораторной находятся в репозитории <https://github.com/mr-lobstr/Laboratory-2.git> .