**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИИ**  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**  
**(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Расчетно-графическое задание

Функциональное и логическое программирование

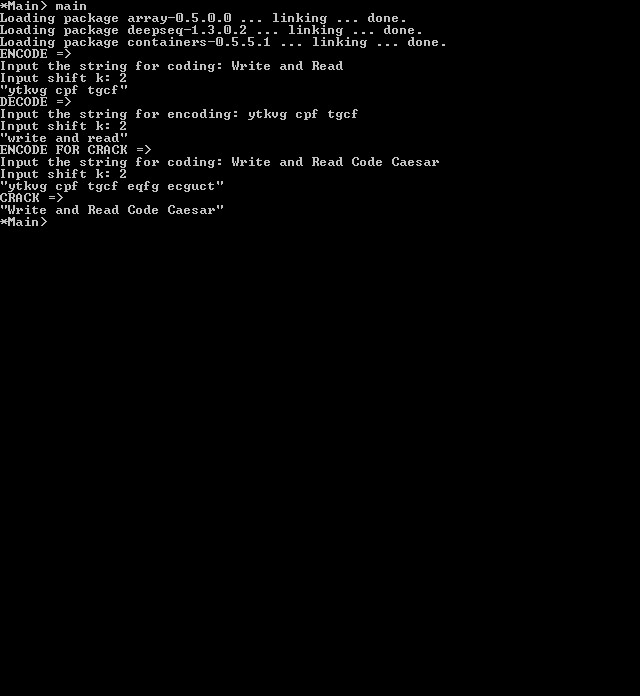
Выполнил: ст. группы ПВ-21  
Ковалев Павел Александрович

Проверил: Поляков В.К.

**Белгород 2020**

**Задания**

Выбраная тема: шифр цезаря



**Код программы**

--Шифр Цезаря

--y=(x+k) mod n - шифрование

--x=(y-k) mod n - дешифрование

--map (a -> b) -> [a] -> [b] возвращает список, созданный путем применения функции (первый аргумент)

--ко всем элементам в списке, переданном в качестве второго аргумента

--chr Int -> Char

--k = 26 мощность английского алфавита

import System.IO

import Data.Char

import Data.Maybe

import Data.List

import qualified Data.Set as Set

main = do

hSetBuffering stdout NoBuffering

putStrLn "ENCODE =>"

putStr "Input the string for coding: "

s<- getLine

putStr "Input shift k: "

a <- readLn

print $ caesar a s

putStrLn "DECODE =>"

putStr "Input the string for encoding: "

s<- getLine

putStr "Input shift k: "

a <- readLn

print $ unCaesar a s

--

putStrLn "ENCODE FOR CRACK =>"

putStr "Input the string for coding: "

s1<- getLine

putStr "Input shift k: "

a1 <- readLn

print $ caesar a1 s1

putStrLn "CRACK =>"

print $ crack s1

caesar :: Int -> String -> String

caesar n = map coded

where

coded ' ' = ' '

coded c = chr $ (ord (toLower c) - base + n) `mod` 26 + base

base = ord 'a'

unCaesar :: Int -> String -> String

unCaesar n = caesar (26 - n)

--Взлом

let2int :: Char -> Int

let2int c = ord c - ord 'a'

int2let :: Int -> Char

int2let n = chr (ord 'a' + n)

shift :: Int -> Char -> Char

shift n c | isLower c = int2let (((let2int c) + n) `mod` 26)

| otherwise = c

encode :: Int -> String -> String

encode n cs = [ shift n c | c <- cs ]

table :: [Float]

table = [8.2, 1.5, 2.8, 4.3, 12.7, 2.2, 2.0, 6.1, 7.0, 0.2, 0.8, 4.0, 2.4,

6.7, 7.5, 1.9, 0.1, 6.0, 6.3, 9.1, 2.8, 1.0, 2.4, 0.2, 2.0, 0.1]

percent :: Int -> Int -> Float

percent n m = (fromIntegral n / fromIntegral m)\*100

freqs :: String -> [Float]

freqs cs = [percent (count c cs) n | c <- ['a'..'z'] ]

where n = lowers cs

chisqr :: [Float] -> [Float] -> Float

chisqr os es = sum [((o-e)^2)/e | (o,e) <- zip os es]

rotate :: Int -> [a] -> [a]

rotate n xs = drop n xs ++ take n xs

lowers :: String -> Int

lowers cs = length [ c | c <- cs, isLower c]

count :: Char -> String -> Int

count c cs = length [ c' | c' <- cs, c==c']

positions :: Eq a => a -> [a] -> [Int]

positions x xs = [ i' | (x', i') <- zip xs [0..n], x==x' ]

where n = length xs - 1

crack :: String -> String

crack cs = encode (-factor) cs

where factor = head (positions (minimum chitab) chitab)

chitab = [ chisqr (rotate n table') table | n <- [0..25] ]

table' = freqs cs