Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЛАБОРАТОРНА РОБОТА № 1

з дисципліни

«Мультипарадигменне програмування»

ВИКОНАВ:
студент III курсу ФІОТ
групи ІО-23
Бичкар Н. В.
Залікова № 2302

ПЕРЕВІРИВ:
ас. Очеретяний О. К.

Київ – 2025

Лістинг:

```
program lab1
   implicit none

   ! Оголошення змінних
   real, allocatable :: numbers(:), original_numbers(:)
   integer :: n, i, j, index, alphabet_size
   character(len=1), allocatable :: alphabet(:)
   integer, allocatable :: transition_matrix(:,:)
   real :: min_val, max_val, lambda
   real, allocatable :: interval_bounds(:)
   character(len=1), allocatable :: sequence(:)
   integer :: row, col

   ! === 1. Задаємо потужність алфавіту ===
   alphabet_size = 10

   ! === 2. Створюємо алфавіт і матрицю ===
   allocate(alphabet(alphabet_size))
   allocate(transition_matrix(alphabet_size, alphabet_size))
   transition_matrix = 0

   do i = 1, alphabet_size
      alphabet(i) = achar(64 + i)  ! 'A', 'B', ...
   end do

   ! === 3. Задаємо кількість чисел та самі дані ===
   n = 100  ! Розмір вхідного ряду

   ! === 4. Виділення пам'яті ===
   allocate(numbers(n))
   allocate(original_numbers(n))
   allocate(sequence(n))
   allocate(interval_bounds(alphabet_size+1))

   ! === 5. Генеруємо тестові дані ===
   ! Використовуємо генератор випадкових чисел для створення даних
   call random_seed()
   do i = 1, n
      call random_number(original_numbers(i))
      original_numbers(i) = original_numbers(i) * 10.0  ! Масштабуємо до діапазону [0, 10]
   end do

   ! Альтернативний варіант - задаємо конкретні значення
   ! Для демонстраційних цілей можна використати цей варіант замість випадкових
чисел
   ! original_numbers = [2.5, 3.7, 1.2, 4.8, 5.1, 2.2, 3.8, 4.3, 1.9, 5.0,
   !                     3.1, 4.2, 2.8, 5.5, 3.4, 1.7, 4.9, 2.3, 3.5, 4.1,
```

```
!                   1.4, 3.3, 5.2, 2.9, 4.7, 3.2, 1.8, 4.5, 2.0, 5.9]

print *, "Кількість згенерованих чисел:", n
print *, "Перші 10 чисел ряду:"
write(*, '(10F7.2)') (original_numbers(i), i = 1, min(10, n))

! === 6. Копія для сортування — numbers ===
numbers = original_numbers
call sort(numbers, n)

! === 7. Побудова інтервалів за Пуассонівським розподілом ===
min_val = numbers(1)
max_val = numbers(n)

! Параметр lambda для Пуассонівського розподілу (середнє значення)
! Для Пуассонівського розподілу lambda повинно бути позитивним
lambda = sum(numbers) / n

! Виконуємо нормалізацію даних для Пуассонівського розподілу
! Якщо дані мають негативні значення, зміщуємо їх у позитивну область
if (min_val < 0.0) then
   original_numbers = original_numbers - min_val + 1.0
   numbers = original_numbers
   call sort(numbers, n)
   min_val = numbers(1)
   max_val = numbers(n)
   lambda = sum(numbers) / n
end if

! Якщо lambda занадто мала для робочого розподілу Пуассона, коригуємо її
if (lambda < 1.0) then
   lambda = 1.0 + lambda  ! Забезпечуємо мінімальне значення lambda
end if

print *, "Lambda (середнє значення):", lambda

! Встановлюємо межі інтервалів згідно Пуассонівського розподілу
call calculate_poisson_intervals(interval_bounds, alphabet_size, min_val, max_val,
lambda)

! === 8. Перетворення чисел у літери (по оригінальному порядку) ===
do i = 1, n
   index = find_interval(original_numbers(i), interval_bounds, alphabet_size)
   sequence(i) = alphabet(index)
end do

! === 9. Побудова матриці передування ===
do i = 1, n - 1
```

```
    row = index_in_alphabet(sequence(i), alphabet, alphabet_size)
    col = index_in_alphabet(sequence(i + 1), alphabet, alphabet_size)
    transition_matrix(row, col) = transition_matrix(row, col) + 1
end do

! === 10. Виведення результатів ===
print *, "================================================"
print *, "Вхідні дані (перші 20 значень):"
write(*, '(10F7.2)') (original_numbers(i), i = 1, min(20, n))

print *, "================================================"
print *, "Параметри розподілу:"
print *, "Alphabet Size:", alphabet_size
print *, "Мінімальне значення:", min_val
print *, "Максимальне значення:", max_val
print *, "Lambda (середнє значення):", lambda

print *, "================================================"
print *, "Інтервали розподілу Пуассона:"
do i = 1, alphabet_size
    print '(A, I2, A, A, A, F7.2, A, F7.2)', "Інтервал ", i, " (", alphabet(i), "): ", &
          interval_bounds(i), " - ", interval_bounds(i+1)
end do

print *, "================================================"
print *, "Linguistic Sequence (перші 50 символів):"
write(*, '(50A2)') (sequence(i), ' ', i = 1, min(50, n))

print *, "================================================"
print *, "Частота символів у послідовності:"
do i = 1, alphabet_size
    j = count(sequence == alphabet(i))
    if (j > 0) then
        print '(A, A, A, I4, A, F6.2, A)', "Символ ", alphabet(i), ": ", j, " (", real(j)/n*100, "%)"
    end if
end do

print *, "================================================"
print *, "Матриця передування (Transition Matrix):"
! Виведення заголовків стовпців
write(*, '(A4)', advance='no') '    '
do j = 1, alphabet_size
    write(*, '(A4)', advance='no') alphabet(j)
end do
write(*, *)

! Виведення матриці з позначеннями рядків
do i = 1, alphabet_size
```

```
        write(*, '(A4)', advance='no') alphabet(i)
        write(*, '(10I4)') (transition_matrix(i, j), j = 1, alphabet_size)
    end do

    ! === 11. Звільнення пам'яті ===
    deallocate(numbers, original_numbers, sequence, alphabet, transition_matrix,
interval_bounds)

contains

    ! === Функція сортування масиву ===
    subroutine sort(arr, size)
        implicit none
        integer, intent(in) :: size
        real, intent(inout) :: arr(size)
        integer :: i, j
        real :: temp

        do i = 1, size-1
            do j = i+1, size
                if (arr(i) > arr(j)) then
                    temp = arr(i)
                    arr(i) = arr(j)
                    arr(j) = temp
                end if
            end do
        end do
    end subroutine sort

    ! === Функція знаходження індексу символу у алфавіті ===
    function index_in_alphabet(c, alph, size) result(indx)
        implicit none
        character(len=1), intent(in) :: c
        character(len=1), intent(in) :: alph(size)
        integer, intent(in) :: size
        integer :: indx, i

        indx = 0
        do i = 1, size
            if (c == alph(i)) then
                indx = i
                exit
            end if
        end do
    end function index_in_alphabet

    ! === Функція для обчислення факторіалу ===
    function factorial(n) result(fact)
```

```
    implicit none
    integer, intent(in) :: n
    real :: fact
    integer :: i

    fact = 1.0
    do i = 2, n
        fact = fact * i
    end do
end function factorial

! === Функція для наближення erf (функція помилок) ===
function my_erf(x) result(result)
    implicit none
    real, intent(in) :: x
    real :: result, t, series
    real, parameter :: a1 = 0.254829592, a2 = -0.284496736, a3 = 1.421413741
    real, parameter :: a4 = -1.453152027, a5 = 1.061405429, p = 0.3275911

    ! Сохранюємо знак x
    if (x >= 0.0) then
        t = 1.0 / (1.0 + p * x)
        series = a1 + t * (a2 + t * (a3 + t * (a4 + t * a5)))
        result = 1.0 - series * exp(-x * x)
    else
        t = 1.0 / (1.0 - p * x)
        series = a1 + t * (a2 + t * (a3 + t * (a4 + t * a5)))
        result = series * exp(-x * x) - 1.0
    end if
end function my_erf

! === Функція для обчислення функції розподілу Пуассона ===
function poisson_cdf(x, lambda) result(cdf)
    implicit none
    real, intent(in) :: x, lambda
    real :: cdf, term, sum_terms
    integer :: i, k, max_k

    if (x < 0.0) then
        cdf = 0.0
        return
    end if

    k = int(x)

    ! Для великих значень lambda використовуємо нормальне наближення
    if (lambda > 20.0) then
        ! Наближення нормальним розподілом для великих lambda
```

```
        ! (x + 0.5 - lambda) / sqrt(lambda)
        cdf = 0.5 + 0.5 * my_erf((x + 0.5 - lambda) / sqrt(2.0 * lambda))
        return
    end if

    ! Для малих значень lambda - обчислюємо суму ряду
    cdf = 0.0
    sum_terms = 0.0
    term = exp(-lambda)  ! перший член ряду (для i=0)

    max_k = min(k, 100)  ! Обмеження для запобігання переповнення

    do i = 0, max_k
        sum_terms = sum_terms + term
        if (i == max_k) exit

        ! Обчислюємо наступний член рекурентно для стабільності
        term = term * lambda / (i + 1)
    end do

    cdf = sum_terms
  end function poisson_cdf

  ! === Функція для обчислення меж інтервалів за Пуассонівським розподілом ===
  subroutine calculate_poisson_intervals(bounds, num_intervals, min_val, max_val,
lambda)
    implicit none
    real, intent(inout) :: bounds(num_intervals+1)
    integer, intent(in) :: num_intervals
    real, intent(in) :: min_val, max_val, lambda
    real :: range, scaled_lambda, x, prob_step, prob_target
    integer :: i

    ! Встановлюємо першу і останню межу
    bounds(1) = min_val
    bounds(num_intervals+1) = max_val

    range = max_val - min_val
    scaled_lambda = lambda  ! Можна скоригувати при необхідності

    ! Крок ймовірності для рівномірного розбиття за ймовірністю
    prob_step = 1.0 / num_intervals

    ! Обчислюємо межі інтервалів на основі квантилів розподілу Пуассона
    do i = 1, num_intervals - 1
        prob_target = i * prob_step

        ! Використовуємо бінарний пошук для знаходження квантиля
```

```
        bounds(i+1) = find_poisson_quantile(prob_target, scaled_lambda, min_val, max_val)
      end do
   end subroutine calculate_poisson_intervals

   ! === Функція для знаходження квантиля розподілу Пуассона методом бінарного пошуку ===
   function find_poisson_quantile(prob, lambda, min_val, max_val) result(quantile)
      implicit none
      real, intent(in) :: prob, lambda, min_val, max_val
      real :: quantile, left, right, mid, cdf_mid, scaled_x
      integer :: iter

      ! Ініціалізуємо межі пошуку
      left = 0.0
      right = lambda * 5.0 + 10.0  ! Верхня межа для Пуассона (зазвичай достатньо lambda*5)

      ! Бінарний пошук для знаходження квантиля
      do iter = 1, 100  ! Максимальна кількість ітерацій
         mid = (left + right) / 2.0
         cdf_mid = poisson_cdf(mid, lambda)

         if (abs(cdf_mid - prob) < 1.0E-6) then
            exit  ! Достатня точність
         else if (cdf_mid < prob) then
            left = mid
         else
            right = mid
         end if

         if (right - left < 1.0E-6) exit  ! Досягнута точність
      end do

      ! Перетворюємо кількісне значення випадкової величини у відповідне значення вхідного ряду
      scaled_x = min_val + (mid / (lambda * 5.0 + 10.0)) * (max_val - min_val)
      quantile = scaled_x
   end function find_poisson_quantile

   ! === Функція для знаходження індексу інтервалу, до якого належить значення ===
   function find_interval(val, bounds, num_intervals) result(idx)
      implicit none
      real, intent(in) :: val
      real, intent(in) :: bounds(num_intervals+1)
      integer, intent(in) :: num_intervals
      integer :: idx, i

      idx = 1  ! За замовчуванням - перший інтервал
```

```
        do i = 1, num_intervals
            if (val >= bounds(i) .and. val < bounds(i+1)) then
                idx = i
                exit
            end if
        end do

        ! Якщо значення дорівнює верхній межі останнього інтервалу
        if (val == bounds(num_intervals+1)) then
            idx = num_intervals
        end if
    end function find_interval

end program lab1
```

Результати:

```
Кількість згенерованих чисел:         100
Перші 10 чисел ряду:
  8.32   9.28   0.99   1.25   4.63   4.97   5.84   8.62   4.47   7.52
Lambda (середнє значення):   5.51422977
================================================
Вхідні дані (перші 20 значень):
  8.32   9.28   0.99   1.25   4.63   4.97   5.84   8.62   4.47   7.52
  6.69   9.80   1.21   8.94   4.25   9.81   4.49   1.32   4.80   9.45
================================================
П?раметри розподілу:
Alphabet Size:          10
Мінімальне значення:   6.95568323E-02
Максимальне значення:   9.92683506
Lambda (середнє значення):   5.51422977
================================================
Інтервали розподілу Пуассона:
Інтервал  1 (A):    0.07 -    0.86
Інтервал  2 (B):    0.86 -    0.86
Інтервал  3 (C):    0.86 -    1.12
Інтервал  4 (D):    1.12 -    1.38
Інтервал  5 (E):    1.38 -    1.38
І?тервал  6 (F):    1.38 -    1.64
Інтервал  7 (G):    1.64 -    1.91
Інтер?ал  8 (H):    1.91 -    1.91
Інтервал  9 (I):    1.91 -    2.43
Інтервал 10 (J):    2.43 -    9.93
================================================
Linguistic Sequence (перші 50 символів):
J   J   C   D   J   J   J   J   J   J   J   J   D   J   J   J   J   D   J   J   J   C   J   J   J
J   J   J   J   J   J   J   J   J   J   J   J   J   J   J   J   I   J   J   F   J   J   J   F
================================================
```

```
==========================================
Частота символів у послідовності:
Символ A:     5 (  5.00%)
Символ C:     2 (  2.00%)
Символ D:     4 (  4.00%)
Символ F:     2 (  2.00%)
Символ G:     1 (  1.00%)
Символ I:     2 (  2.00%)
Символ J:    84 ( 84.00%)
==========================================
Матриця передування (Transition Matrix):
      A   B   C   D   E   F   G   H   I   J
  A   0   0   0   0   0   0   0   0   0   5
  B   0   0   0   0   0   0   0   0   0   0
  C   0   0   0   1   0   0   0   0   0   1
  D   0   0   0   0   0   0   0   0   0   4
  E   0   0   0   0   0   0   0   0   0   0
  F   0   0   0   0   0   0   0   0   0   2
  G   0   0   0   0   0   0   0   0   0   1
  H   0   0   0   0   0   0   0   0   0   0
  I   0   0   0   0   0   0   0   0   0   2
  J   5   0   2   3   0   2   1   0   2  68


...Program finished with exit code 0
Press ENTER to exit console.
```