Pro 1a:

```
pro_1a = function() {
  # Function to find the GCD using Euclidean algorithm
  gcd = function(a, b) {
    while (b != 0) {
      temp = b
      b = a %% b
      a = temp
    }
    return(a)
  }

  # Function to find the LCM using the GCD
  lcm = function(a, b) {
    return(abs(a * b) / gcd(a, b))
  }

  # Example usage
  a = as.integer(readline(prompt="Enter first number: "))
  b = as.integer(readline(prompt="Enter second number: "))
  cat("LCM of", a, "and", b, "is:", lcm(a, b), "\n")
}

# Calling the function
pro_1a()
```

Output:

```
Enter first number: 5
Enter second number: 10
LCM of 5 and 10 is: 10
```

Pro1b:

```
pro_1b = function() {
  # Function to calculate mean, variance, and standard deviation
  calculate_statistics = function(values, probabilities) {
    mean = sum(values * probabilities)
    variance = sum((values - mean)^2 * probabilities)
    std_dev = sqrt(variance)
```

```
    return(list(mean = mean, variance = variance, std_dev = std_dev))
  }

  # Example usage
  values = c(-2, -1, 0, 1)
  probabilities = c(1/8, 1/8, 1/4, 1/2)

  stats = calculate_statistics(values, probabilities)
  cat("Mean:", stats$mean, "\n")
  cat("Variance:", stats$variance, "\n")
  cat("Standard Deviation:", stats$std_dev, "\n")
}

# Calling the function
pro_1b()
```

Output:
Mean: 0.125
Variance: 1.109375
Standard Deviation: 1.053269

<u>Pro 2a:</u>

```
pro_2a = function() {
  is_prime = function(n) {
    if (n <= 1) return(FALSE)
    if (n <= 3) return(TRUE)
    if (n %% 2 == 0 || n %% 3 == 0) return(FALSE)
    i = 5
    while (i * i <= n) {
      if (n %% i == 0 || n %% (i + 2) == 0) return(FALSE)
      i = i + 6
    }
    return(TRUE)
  }

  # Example usage
  n = as.integer(readline(prompt="Enter a number: "))
  if (is_prime(n)) {
    cat(n, "is a prime number.\n")
  } else {
```

```
    cat(n, "is not a prime number.\n")
  }
}

# Calling the function
pro_2a()
```

Output:
Enter a number: 5
5 is a prime number.

<u>Pro 2b:</u>

```
pro_2b = function() {
  mean = 3
  variance = 9/4
  p = 1 - sqrt(1 - 4 * variance / mean^2)
  n = mean / p

  prob = dbinom(0:7, size = n, prob = p)
  P_X_le_7 = sum(prob[1:8])
  P_1_le_X_lt_6 = sum(prob[2:6])

  cat("P(X ≤ 7):", P_X_le_7, "\n")
  cat("P(1 ≤ X < 6):", P_1_le_X_lt_6, "\n")
}

# Calling the function
pro_2b()
```

Output:
P(X ≤ 7): 1
P(1 ≤ X < 6): 1

<u>Pro 3a:</u>

```
pro3a = function(){
# Define the mean and standard deviation
mean_x <- 30
sd_x <- 5

# 1. P(26 <= X <= 40)
```

```
prob_26_to_40 <- pnorm(40, mean = mean_x, sd = sd_x) - pnorm(26, mean = mean_x,
sd = sd_x)
cat("P(26 <= X <= 40) =", prob_26_to_40, "\n")

# 2. P(X >= 45)
prob_45_or_more <- 1 - pnorm(45, mean = mean_x, sd = sd_x)
cat("P(X >= 45) =", prob_45_or_more, "\n")
}
pro3a()
```

Output:
P(26 <= X <= 40) = 0.7653945
P(X >= 45) = 0.001349898

Pro 3b:

```
pro_3b = function() {
  # Function to calculate mean, median, and mode
  calculate_statistics = function(data) {
    mean = mean(data)
    median = median(data)
    mode = as.numeric(names(sort(table(data), decreasing=TRUE)[1]))

    return(list(mean = mean, median = median, mode = mode))
  }

  # Example usage
  data = as.numeric(strsplit(readline(prompt="Enter the data (space-separated): "), "
")[[1]])
  stats = calculate_statistics(data)
  cat("Mean:", stats$mean, "\n")
  cat("Median:", stats$median, "\n")
  cat("Mode:", stats$mode, "\n")
}

# Calling the function
pro_3b()
```

Output:
Enter the data (space-separated): 5 10 15 10 25 80
Mean: 24.16667

Median: 12.5
Mode: 10

<u>Pro 4a;</u>

```
pro_4a = function() {
  # Function to find the correlation coefficient
  calculate_correlation = function(x, y) {
    cor(x, y)
  }

  # Example data
  x = c(22, 26, 29, 30, 31, 31, 34, 35)
  y = c(20, 20, 21, 29, 27, 24, 27, 31)
  correlation = calculate_correlation(x, y)
  cat("Correlation coefficient:", correlation, "\n")
}

# Calling the function
pro_4a()
```

Output:
Correlation coefficient: 0.8208434

<u>Pro 4b:</u>

```
pro_4b = function() {
  # Function to perform matrix addition and multiplication
  matrix_operations = function(a, b) {
    addition = a + b
    multiplication = a %*% b
    return(list(addition = addition, multiplication = multiplication))
  }

  # Example usage
  cat("Enter elements of first matrix (space-separated, row-wise):\n")
  a = matrix(as.numeric(strsplit(readline(), " ")[[1]]), nrow=2, byrow=TRUE)
  cat("Enter elements of second matrix (space-separated, row-wise):\n")
  b = matrix(as.numeric(strsplit(readline(), " ")[[1]]), nrow=2, byrow=TRUE)

  results = matrix_operations(a, b)
```

```
  cat("Matrix Addition:\n")
  print(results$addition)
  cat("Matrix Multiplication:\n")
  print(results$multiplication)
}

# Calling the function
pro_4b()
```

Output:
Enter elements of first matrix (space-separated, row-wise):
1 2 3
Enter elements of second matrix (space-separated, row-wise):
4 5 6
Matrix Addition:

```
     [,1] [,2]
[1,]    5    7
[2,]    9    5
```

Matrix Multiplication:

```
     [,1] [,2]
[1,]   16   13
[2,]   18   19
```

Pro 5a:

```
pro_5a = function() {
  # Define the probability distribution
  a = 1/81
  x = 0:8
  p_x = c(a, 3*a, 5*a, 7*a, 9*a, 11*a, 13*a, 15*a, 17*a)

  P_X_less_3 = sum(p_x[1:3])
  P_X_ge_3 = sum(p_x[4:9])
  P_X_between_0_5 = sum(p_x[2:6])

  cat("P(X < 3):", P_X_less_3, "\n")
  cat("P(X ≥ 3):", P_X_ge_3, "\n")
  cat("P(0 < X < 5):", P_X_between_0_5, "\n")
}

# Calling the function
```

```
pro_5a()
```

Output:
P(X < 3): 0.1111111
P(X ≥ 3): 0.8888889
P(0 < X < 5): 0.4320988

Pro 5b:

```
pro_5b = function() {
  # Function to create a data frame
  create_data_frame = function() {
    rno = as.integer(readline(prompt="Enter roll number: "))
    name = readline(prompt="Enter name: ")
    age = as.integer(readline(prompt="Enter age: "))
    blood_group = readline(prompt="Enter blood group: ")
    grade = readline(prompt="Enter grade: ")

    data = data.frame(Roll_No=rno, Name=name, Age=age, Blood_Group=blood_group,
Grade=grade)
    return(data)
  }

  # Create data frame for 5 students
  students = do.call(rbind, replicate(5, create_data_frame(), simplify=FALSE))
  print(summary(students))
  print(str(students))
}

# Calling the function
pro_5b()
```

Output:

```
Enter roll number: 12
Enter name: dncjd
Enter age: 12
Enter blood group: o
Enter grade: a
Enter roll number: 21
Enter name: djjbd
Enter age: 14
```

```
Enter blood group: b
Enter grade: a
Enter roll number: 69
Enter name: dbhvb
Enter age: 54
Enter blood group: b
Enter grade: a
Enter roll number: 89
Enter name: dbch
Enter age: 25
Enter blood group: b+
Enter grade: a
Enter roll number: 87
Enter name: dnhbc
Enter age: 40
Enter blood group: a
Enter grade: c
    Roll_No         Name                Age      Blood_Group
 Min.   :12.0   Length:5           Min.   :12   Length:5
 1st Qu.:21.0   Class :character   1st Qu.:14   Class :character
 Median :69.0   Mode  :character   Median :25   Mode  :character
 Mean   :55.6                      Mean   :29
 3rd Qu.:87.0                      3rd Qu.:40
 Max.   :89.0                      Max.   :54
    Grade
 Length:5
 Class :character
 Mode  :character



'data.frame':   5 obs. of  5 variables:
 $ Roll_No    : int  12 21 69 89 87
 $ Name       : chr  "dncjd" "djjbd" "dbhvb" "dbch" ...
 $ Age        : int  12 14 54 25 40
 $ Blood_Group: chr  "o" "b" "b" "b+" ...
 $ Grade      : chr  "a" "a" "a" "a" ...
```

Pro 6a:

```
pro_6a = function() {
  # Function to find the correlation coefficient
  calculate_correlation = function(x, y) {
    cor(x, y)
  }

  # Example data
  x = c(65, 66, 67, 67, 68, 69, 70, 72)
  y = c(67, 68, 65, 68, 72, 72, 69, 71)
  correlation = calculate_correlation(x, y)
  cat("Correlation coefficient:", correlation, "\n")
}

# Calling the function
pro_6a()
```

Output:

Correlation coefficient: 0.6030227

Pro 6b:

```
pro_6b = function() {
  # Function to find the roots of a quadratic equation
  find_roots = function(a, b, c) {
    discriminant = b^2 - 4*a*c
    if (discriminant > 0) {
      root1 = (-b + sqrt(discriminant)) / (2*a)
      root2 = (-b - sqrt(discriminant)) / (2*a)
      return(c(root1, root2))
    } else if (discriminant == 0) {
      root = -b / (2*a)
      return(root)
    } else {
      return("No real roots")
    }
  }

  # Example usage
  a = as.numeric(readline(prompt="Enter coefficient a: "))
  b = as.numeric(readline(prompt="Enter coefficient b: "))
```

```
  c = as.numeric(readline(prompt="Enter coefficient c: "))

  roots = find_roots(a, b, c)
  cat("Roots of the quadratic equation are:", roots, "\n")
}

# Calling the function
pro_6b()
```

Output:
Enter coefficient a: 1
Enter coefficient b: -5
Enter coefficient c: 6
Roots of the quadratic equation are: 3 2

<u>Pro 7a:</u>

```
pro_7a = function() {
  # Define the probability function
  k = 1 / (1 + 8*2 + 2*3 + 2*4 + 3*5 + 5*6 + 7*7)
  x = 0:7
  p_x = c(0, k, 2*k, 2*k, 3*k, k^2, 2*k^2, 7*k^2 + k)

  P_X_less_6 = sum(p_x[1:6])
  P_X_ge_6 = sum(p_x[7:8])
  P_X_between_0_5 = sum(p_x[2:6])
  P_X_between_0_4 = sum(p_x[1:5])

  cat("P(X < 6):", P_X_less_6, "\n")
  cat("P(X ≥ 6):", P_X_ge_6, "\n")
  cat("P(0 < X < 5):", P_X_between_0_5, "\n")
  cat("P(0 ≤ X ≤ 4):", P_X_between_0_4, "\n")
}

# Calling the function
pro_7a()
```

Output:
P(X < 6): 0.064064
P(X ≥ 6): 0.008576
P(0 < X < 5): 0.064064

P(0 ≤ X ≤ 4): 0.064

Pro 7b:

```
pro_7b = function() {
  # Function to create and access elements in a vector
  create_and_access_vector = function() {
    vec = as.numeric(strsplit(readline(prompt="Enter the elements of the vector
(space-separated): "), " ")[[1]])
    cat("The vector is:", vec, "\n")

    # Access elements
    cat("First element:", vec[1], "\n")
    cat("Last element:", vec[length(vec)], "\n")
    cat("Second to fourth elements:", vec[2:4], "\n")
  }

  # Example usage
  create_and_access_vector()
}

# Calling the function
pro_7b()
```

Output:

```
Enter the elements of the vector (space-separated): 5 10 15 20 25
The vector is: 5 10 15 20 25
First element: 5
Last element: 25
Second to fourth elements: 10 15 20
```

Pro 8a:

```
pro_8a = function() {
  # Function to calculate binomial probabilities
  calculate_binomial_probabilities = function(n, p, k) {
    dbinom(k, size=n, prob=p)
  }

  # Example usage
  n = 18
  p = 0.5
```

```
  k1 = 10
  k2 = 0:18

  P_exactly_10 = calculate_binomial_probabilities(n, p, k1)
  P_at_least_10 = sum(dbinom(10:18, size=n, prob=p))
  P_at_most_8 = sum(dbinom(0:8, size=n, prob=p))

  cat("P(Exactly 10):", P_exactly_10, "\n")
  cat("P(At least 10):", P_at_least_10, "\n")
  cat("P(At most 8):", P_at_most_8, "\n")
}

# Calling the function
pro_8a()
```

Output:
P(Exactly 10): 0.1669235
P(At least 10): 0.4072647
P(At most 8): 0.4072647

<u>Pro 8b:</u>

```
pro_8b = function() {
  # Function to calculate probabilities with two dice
  calculate_dice_probabilities = function() {
    p = 6 / 36
    n = 5
    k1 = 1
    k2 = 2
    k3 = 1:4

    P_at_least_once = sum(dbinom(1:5, size=n, prob=p))
    P_two_times = dbinom(k2, size=n, prob=p)
    P_between_1_and_5 = sum(dbinom(k3, size=n, prob=p))

    cat("P(At least once):", P_at_least_once, "\n")
    cat("P(Two times):", P_two_times, "\n")
    cat("P(1 < X < 5):", P_between_1_and_5, "\n")
  }

  # Example usage
```

```
  calculate_dice_probabilities()
}

# Calling the function
pro_8b()
```

Output:
P(At least once): 0.5981224
P(Two times): 0.160751
P(1 < X < 5): 0.5979938

Pro 9a:

```
pro9a = function(){
# Parameters
mean_height <- 153
std_dev <- 20
total_students <- 100

# Heights of interest
height_lower <- 150
height_upper <- 170

# Calculate z-scores
z_lower <- (height_lower - mean_height) / std_dev
z_upper <- (height_upper - mean_height) / std_dev

# Calculate cumulative probabilities using pnorm
p_lower <- pnorm(z_lower)
p_upper <- pnorm(z_upper)

# Probability of height between 150 cm and 170 cm
probability_between <- p_upper - p_lower

# Number of students with height between 150 cm and 170 cm
number_of_students_between <- probability_between * total_students

# Print the result
print(number_of_students_between)
}
pro9a()
```

Output:
[1] 36.19551

Pro 9b:
```
pro9b = function(){
n <- 5 # number of trials
p <- 1/6 # probability of success on each trial

# (i) Probability of getting a sum of 7 at least once
# P(X >= 1) = 1 - P(X = 0)
prob_at_least_once <- 1 - dbinom(0, n, p)

# (ii) Probability of getting a sum of 7 exactly two times
prob_exactly_two <- dbinom(2, n, p)

# (iii) Probability of getting a sum of 7 between 1 and 4 times (exclusive)
# P(1 < X < 5) = P(X = 2) + P(X = 3) + P(X = 4)
prob_between_1_and_5 <- sum(dbinom(2:4, n, p))

# Display the results
cat("Probability of getting a sum of 7 at least once: ", prob_at_least_once, "\n")
cat("Probability of getting a sum of 7 exactly two times: ", prob_exactly_two, "\n")
cat("Probability of getting a sum of 7 between 1 and 4 times (exclusive): ",
prob_between_1_and_5, "\n")
}
pro9b()
```

Output:
Probability of getting a sum of 7 at least once: 0.5981224
Probability of getting a sum of 7 exactly two times: 0.160751
Probability of getting a sum of 7 between 1 and 4 times (exclusive): 0.1961163

Pro 10a:
```
pro_10a = function() {
  # Function to calculate binomial probabilities for coin tosses
  calculate_coin_probabilities = function(n, p, k) {
    sum(dbinom(k, size=n, prob=p))
  }

  # Example usage
```

```
  n = 10
  p = 0.5

  P_at_least_7_heads = calculate_coin_probabilities(n, p, 7:10)
  P_at_least_6_heads = calculate_coin_probabilities(n, p, 6:10)

  cat("P(At least 7 heads):", P_at_least_7_heads, "\n")
  cat("P(At least 6 heads):", P_at_least_6_heads, "\n")
}

# Calling the function
pro_10a()
```

Output:
P(At least 7 heads): 0.171875
P(At least 6 heads): 0.3769531

<u>Pro 10b:</u>

```
pro_10b = function() {
  # Function to perform matrix operations
  matrix_operations = function() {
    cat("Enter elements of first matrix (space-separated, row-wise):\n")
    a = matrix(as.numeric(strsplit(readline(), " ")[[1]]), nrow=2, byrow=TRUE)
    cat("Enter elements of second matrix (space-separated, row-wise):\n")
    b = matrix(as.numeric(strsplit(readline(), " ")[[1]]), nrow=2, byrow=TRUE)

    addition = a + b
    subtraction = a - b
    multiplication = a %*% b
    transpose_a = t(a)
    transpose_b = t(b)

    cat("Matrix Addition:\n")
    print(addition)
    cat("Matrix Subtraction:\n")
    print(subtraction)
    cat("Matrix Multiplication:\n")
    print(multiplication)
    cat("Transpose of first matrix:\n")
    print(transpose_a)
```

```
    cat("Transpose of second matrix:\n")
    print(transpose_b)
  }

  # Example usage
  matrix_operations()
}

# Calling the function
pro_10b()
```

Output:

```
Enter elements of first matrix (space-separated, row-wise):
1 2 3
Enter elements of second matrix (space-separated, row-wise):
4 5 6
Matrix Addition:
     [,1] [,2]
[1,]    5    7
[2,]    9    5
Matrix Subtraction:
     [,1] [,2]
[1,]   -3   -3
[2,]   -3   -3
Matrix Multiplication:
     [,1] [,2]
[1,]   16   13
[2,]   18   19
Transpose of first matrix:
     [,1] [,2]
[1,]    1    3
[2,]    2    1
Transpose of second matrix:
     [,1] [,2]
[1,]    4    6
[2,]    5    4
```

Pro 11a:

```
pro_11a = function() {
  # Function to calculate binomial probabilities
```

```
  calculate_binomial_probabilities = function(n, p) {
    mean = 3
    variance = 9 / 4
    q = 1 - p
    k_values = 0:n
    binomial_probs = dbinom(k_values, size=n, prob=p)
    return(binomial_probs)
  }

  # Given data
  n = 4  # Derived from mean = np and variance = npq
  p = 3 / 4  # Derived from mean = 3
  probs = calculate_binomial_probabilities(n, p)

  cat("Binomial probabilities:\n")
  print(probs)
}

# Calling the function
pro_11a()
```

Output:
Binomial probabilities:
[1] 0.00390625 0.04687500 0.21093750 0.42187500 0.31640625

<u>Pro 11b</u>:

```
pro_11b = function() {
  # Function to calculate Poisson probabilities
  poisson_probabilities = function(lambda, k) {
    exp(-lambda) * (lambda^k) / factorial(k)
  }

  # Given data
  lambda = 1.5

  # Calculate probabilities
  P_no_demand = poisson_probabilities(lambda, 0)
  P_demand_greater_than_2 = 1 - (poisson_probabilities(lambda, 0) +
poisson_probabilities(lambda, 1) + poisson_probabilities(lambda, 2))
```

```
  cat("Proportion of days with no demand:", P_no_demand, "\n")
  cat("Proportion of days with demand greater than two:", P_demand_greater_than_2,
"\n")
}

# Calling the function
pro_11b()
```

Output:
Proportion of days with no demand: 0.2231302
Proportion of days with demand greater than two: 0.1911532

Pro 12a:

```
pro_12a = function() {
  # Function to calculate the probability using Poisson distribution
  probability_defective = function(lambda, k) {
    exp(-lambda) * (lambda^k) / factorial(k)
  }

  # Given data
  lambda = 1  # For 1% defective in 100 condensers, λ = 100 * 0.01
  n = 100

  # Calculate probabilities
  P_zero_defective = probability_defective(lambda, 0)
  P_one_or_more_defective = 1 - P_zero_defective

  cat("Probability of one or more defective condensers:", P_one_or_more_defective,
"\n")
}

# Calling the function
pro_12a()
```

Output
probability of one or more defective condensers: 0.6321206

Pro 12b:

```
pro_12b = function() {
  # Function to find the value of 'a'
```

```
  find_a = function() {
    mean = 50
    x = c(10, 30, 50, 70, 90)
    f = c(17, 5*a + 3, 32, 7*a - 11, 19)

    # Sum of frequencies
    sum_f = sum(f)

    # Mean formula
    sum_fx = sum(x * f)
    mean_calculated = sum_fx / sum_f

    # Solve for 'a'
    a = (mean * sum_f - sum_fx) / (sum(f) - sum(x))
    return(a)
  }

  # Calculate 'a' and frequencies
  a = find_a()
  f_30 = 5*a + 3
  f_70 = 7*a - 11

  cat("Value of 'a':", a, "\n")
  cat("Frequency of 30:", f_30, "\n")
  cat("Frequency of 70:", f_70, "\n")
}

# Calling the function
pro_12b()
```

Output:

Pro 13a:

```
pro_13a = function() {
  # Function to calculate Poisson probabilities without using built-in functions
  poisson_probabilities_manual = function(lambda, k) {
    exp(-lambda) * (lambda^k) / factorial(k)
  }

  # Using built-in function
```

```
  poisson_probabilities_builtin = function(lambda, k) {
    ppois(k, lambda)
  }

  # Given data
  lambda = 1.5

  # Calculate probabilities without built-in functions
  P_no_demand_manual = poisson_probabilities_manual(lambda, 0)
  P_demand_greater_than_2_manual = 1 - (poisson_probabilities_manual(lambda, 0) +
poisson_probabilities_manual(lambda, 1) + poisson_probabilities_manual(lambda, 2))

  # Calculate probabilities with built-in functions
  P_no_demand_builtin = poisson_probabilities_builtin(lambda, 0)
  P_demand_greater_than_2_builtin = 1 - ppois(2, lambda)

  cat("Proportion of days with no demand (manual):", P_no_demand_manual, "\n")
  cat("Proportion of days with demand greater than two (manual):",
P_demand_greater_than_2_manual, "\n")
  cat("Proportion of days with no demand (built-in):", P_no_demand_builtin, "\n")
  cat("Proportion of days with demand greater than two (built-in):",
P_demand_greater_than_2_builtin, "\n")
}

# Calling the function
pro_13a()
```

Output:
Proportion of days with no demand (manual): 0.2231302
Proportion of days with demand greater than two (manual): 0.1911532
Proportion of days with no demand (built-in): 0.2231302
Proportion of days with demand greater than two (built-in): 0.1911532

<u>Pro 13b:</u>

```
pro_13b = function() {
  # Function to calculate mean salary
  calculate_mean_salary = function(salaries, workers) {
    total_salary = sum(salaries * workers)
    total_workers = sum(workers)
    mean_salary = total_salary / total_workers
```

```
    return(mean_salary)
  }

  # Given data
  salaries = c(3, 4, 5, 6, 7, 8, 9, 10)
  workers = c(16, 12, 10, 8, 6, 4, 3, 1)

  mean_salary = calculate_mean_salary(salaries, workers)
  cat("Mean salary of workers:", mean_salary, "\n")
}

# Calling the function
pro_13b()
```

Output:
Mean salary of workers: 5.083333

Pro 14a:

```
pro_14a = function() {
  # Function to calculate mean, median, and mode
  calculate_statistics = function(data) {
    n = length(data)
    mean = sum(data) / n
    sorted_data = sort(data)
    median = if (n %% 2 == 0) {
      (sorted_data[n/2] + sorted_data[n/2 + 1]) / 2
    } else {
      sorted_data[(n + 1) / 2]
    }
    mode = as.numeric(names(sort(table(data), decreasing=TRUE)[1]))

    return(list(mean = mean, median = median, mode = mode))
  }

  # Given data
  data = c(87, 71, 83, 67, 85, 77, 69, 76, 65, 85, 85, 54, 70, 68, 80, 73, 78, 68, 85, 73,
81, 78, 81, 77, 75)

  stats = calculate_statistics(data)
  cat("Mean:", stats$mean, "\n")
```

```
  cat("Median:", stats$median, "\n")
  cat("Mode:", stats$mode, "\n")
}

# Calling the function
pro_14a()
```

Output:
Mean: 75.64
Median: 77
Mode: 85

Pro 14b:

```
pro_14b = function() {
  # Function to find L.C.M using Euclidean algorithm
  gcd = function(a, b) {
    while (b != 0) {
      temp = b
      b = a %% b
      a = temp
    }
    return(a)
  }

  lcm = function(a, b) {
    return((a * b) / gcd(a, b))
  }

  # Taking input from user
  a = as.integer(readline(prompt="Enter the first number: "))
  b = as.integer(readline(prompt="Enter the second number: "))

  result = lcm(a, b)
  cat("L.C.M of", a, "and", b, "is:", result, "\n")
}

# Calling the function
pro_14b()
```

Output:

Enter the first number: 5
Enter the second number: 10
L.C.M of 5 and 10 is: 10

Pro 15a:

```
pro_15a = function() {
  # Function to calculate mean for the given distribution
  calculate_mean = function(intervals, frequencies) {
    midpoints = (intervals[,1] + intervals[,2]) / 2
    total_frequency = sum(frequencies)
    mean = sum(midpoints * frequencies) / total_frequency
    return(mean)
  }

  # Given data
  intervals = matrix(c(10, 25, 25, 40, 40, 55, 55, 70, 70, 85, 85, 100), ncol=2,
byrow=TRUE)
  frequencies = c(2, 3, 7, 6, 6, 6)

  mean = calculate_mean(intervals, frequencies)
  cat("Mean for the given distribution:", mean, "\n")
}

# Calling the function
pro_15a()
```

Output:
Mean for the given distribution: 62

Pro 15b:

```
pro_15b = function() {
  # Function to calculate mean and variance
  calculate_mean_variance = function() {
    sides = 6
    outcomes = 2:sides + sides
    probabilities = rep(1 / sides, sides)

    mean = sum(outcomes * probabilities)
    variance = sum((outcomes - mean)^2 * probabilities)
```

```
    return(list(mean = mean, variance = variance))
  }

  # Calculate mean and variance
  stats = calculate_mean_variance()
  cat("Mean:", stats$mean, "\n")
  cat("Variance:", stats$variance, "\n")
}

# Calling the function
pro_15b()
```

Output:
Mean: 9.666667
Variance: 2.222222