Lab Session 2:

Getting familiar with the C programming syntax, programming environment, introduction to variable types and string input output

1. Type conversion

Implicit Type Conversion: When variables are promoted to higher precision, data is preserved. This is automatically done by the compiler for mixed data type expressions.

Explicit Type Conversion (Type Casting): This is performed explicitly by the programmer using casting operators.

The following exercises are designed to provide familiarity and hands on understanding of type conversions.

Exercise 1.1: Implicit type conversion

Write a C program that takes two integers as input from the user, and calculates the average of two integers. Ensure that the result is a floating-point number. Print the result with a floating point precision of 2 (display only 2 decimal places).

Exercise 1.2: Explicit type conversion

Write a C program that take a temperature in Celsius from the user, and converts it into Fahrenheit. Prompt the user to enter the temperature in Celsius as a floating-point number, perform the conversion, and display the result as an integer (rounded to the nearest degree Fahrenheit).

2. ASCII value

ASCII (American Standard Code for Information Interchange) is a fundamental character encoding standard used in computers and digital communication systems. This assigns a unique numeric value (ASCII code) to each character, allowing computers to represent and process text-based information. In ASCII, characters are represented by integers. Each character corresponds to a specific numeric code.

Example:

The ASCII code for the letter 'A' is 65.

The ASCII code for the letter 'a' is 97.

The ASCII code for the digit '0' is 48.

Exercise 2.1:

a) Write a C program that takes a character as input, type cast it into an integer(ASCII value) and print the character and its ASCII value.

b) Extend this C program to now also take an ASCII value (1-255), type cast it into a character and print the ASCII value and its character.

Exercise 2.2:

Write a program to take two variables (one of character type and one of integer type) as input, add them and print the sum both in integer and ASCII value.

3. Pre-increment and post-increment operators

Pre-increment (++i): the variable is incremented by 1 before its value is used in an expression.

Post-increment (i++): the current value of the variable is used in an expression, and then it is incremented by 1.

Exercise 3.1:

Write a C program that initializes an integer variable with a value and demonstrates the difference between pre-increment and post-increment by using both operators on the variable. Print the values before and after the increment operations.

a) Observe how the increment operation is performed. Update the program to use the decrement operator instead. Experiment with also first doing pre and then post and vice versa. Observe the values in both cases.

4. Relational and logic operators:

Exercise 4.1:

Imagine you are programming a robot explorer that is navigating through a space filled with obstacles. The robot is equipped with a distance sensor that measures the distance to the nearest obstacle in front of it.

For the sake of simplicity, take the distance to the nearest obstacle (name this variable as *distanceToObstacle*) as input from the user.

Your task is to create a C program that makes real-time decisions for the robot based on the sensor data.

Implement decision logic using "if" statements and relational operators to guide the robot's actions based on the distance sensor reading entered by the user:

If distanceToObstacle is less than or equal to 20 cm, recommend stopping the robot to avoid a collision.

If distanceToObstacle is between 21 cm and 50 cm, recommend slowing down and proceeding with caution.

If distanceToObstacle is greater than 50 cm, recommend maintaining the current speed.

5. Precedence and Order of Evaluation

Exercise 5.1:

Consider the following expressions:

i. **x = (a + b) * c / d - e;**
ii. **y = a++ + ++b * c / d - e--;**

Your task is to predict the final values of **x** and **y** based on the given expressions. Assume that **a**, **b**, **c**, **d**, and **e** are integer variables, and they have the following initial values:

a = 5

b = 3

c = 4

d = 2

e = 7

Evaluate **i** and **ii** step by step, considering operator precedence and associativity.

Keep track of the intermediate values of the variables as you evaluate the expressions.

Determine the final values of x and y.

Now write a C program to run the two expressions and display their output to verify the step-by-step evaluation that you performed.