|  |  |
| --- | --- |
|  | **AIR UNIVERSITY** |
| **DEPARTMENT OF COMPUTER SCIENCE** |
| **Lab Task 9** |

**Student Name: Hamza Umer Farooq Reg. No: 200789**

**Subject: Compiler Construction Semester: VIII**

**Objective: Productions**

**ASSESSMENT:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attributes** | **Excellent**  **(5)** | **Good**  **(4)** | **Average**  **(3)** | **Satisfactory**  **(2)** | **Unsatisfactory (1)** |
| **Ability to Conduct**  Task |  |  |  |  |  |
| **Ability to assimilate the results** |  |  |  |  |  |
| **Effective use of theorems/postulates/formulas** |  |  |  |  |  |

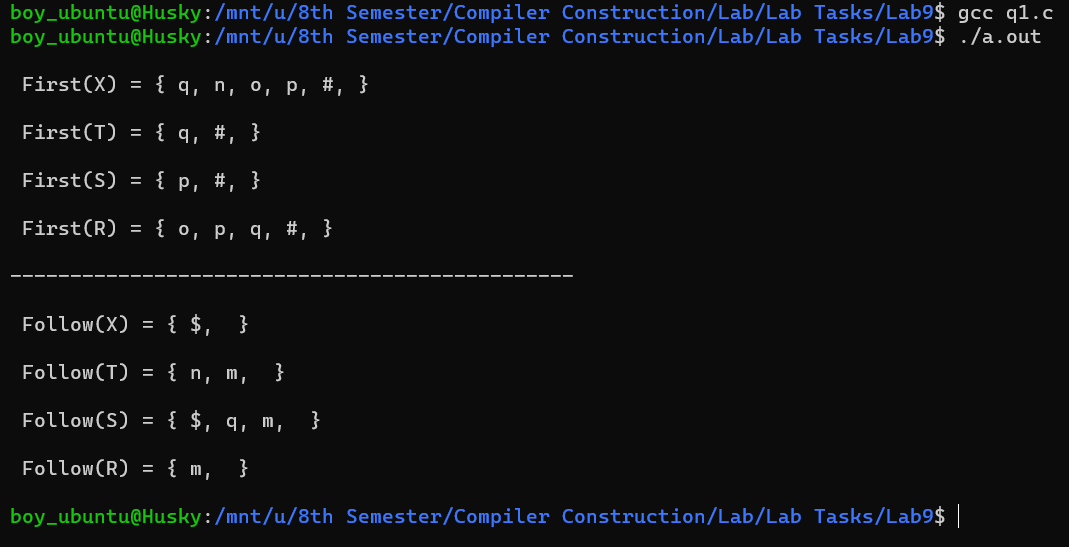
Total Marks:

Obtained Marks:

**REPORT ASSESSMENT:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attributes** | **Excellent**  **(5)** | **Good**  **(4)** | **Average**  **(3)** | **Satisfactory**  **(2)** | **Unsatisfactory**  **(1)** |
| **Data presentation** |  |  |  |  |  |
| **Experimental results** |  |  |  |  |  |
| **Conclusion** |  |  |  |  |  |

Q1.c



Task.c

#include <stdio.h>

#include <ctype.h>

// Function prototypes

void FIRST(char[], char);

void FOLLOW(char[], char);

void addToResultSet(char[], char);

void followfirst(char, int, int, char[]);

// Global variables

int numOfProductions;

char productionSet[10][10];

// Main function

int main()

{

    int i, j;

    char c;

    char result[20];

    // User input for number of productions

    printf("How many number of productions? : ");

    scanf(" %d", &numOfProductions);

    // Input productions

    for (i = 0; i < numOfProductions; i++)

    {

        printf("Enter production Number %d : ", i + 1);

        scanf(" %s", productionSet[i]);

    }

    // Calculate and display FIRST sets

    printf("FIRST OF:\n");

    for (i = 0; i < numOfProductions; i++)

    {

        c = productionSet[i][0];

        FIRST(result, c);

        if (productionSet[i][2] != '$')

        {

            printf("(%c) = {", c);

            for (j = 0; result[j] != '\0'; j++)

            {

                printf("%c ", result[j]);

            }

            printf("}\n");

        }

    }

    // Calculate and display FOLLOW sets

    printf("\nFOLLOW OF:\n");

    for (i = 0; i < numOfProductions; i++)

    {

        c = productionSet[i][0];

        FOLLOW(result, c);

        if (productionSet[i][2] != '$')

        {

            printf("(%c) = {", c);

            for (j = 0; result[j] != '\0'; j++)

            {

                printf("%c ", result[j]);

            }

            printf("}\n");

        }

    }

    return 0;

}

// Function to calculate FIRST set

void FIRST(char \*Result, char c)

{

    int i, j;

    // Initialize Result array

    Result[0] = '\0';

    // If the symbol is a terminal, add it to the FIRST set

    if (!isupper(c)) {

        addToResultSet(Result, c);

    } else {

        // Iterate over productions

        for (i = 0; i < numOfProductions; i++) {

            // Find production with given Non-Terminal symbol

            if (productionSet[i][0] == c) {

                // If production has epsilon, add it to FIRST set

                if (productionSet[i][2] == '$') {

                    addToResultSet(Result, '$');

                } else {

                    // Iterate over symbols in the production

                    j = 2;

                    while (productionSet[i][j] != '\0') {

                        // Calculate FIRST of symbols in the production

                        if (!isupper(productionSet[i][j])) {

                            addToResultSet(Result, productionSet[i][j]);

                            break;

                        } else {

                            // Recursively calculate FIRST of Non-Terminal symbols

                            char subResult[20];

                            FIRST(subResult, productionSet[i][j]);

                            // Add FIRST set to Result set

                            for (int k = 0; subResult[k] != '\0'; k++) {

                                if (subResult[k] != '$') {

                                    addToResultSet(Result, subResult[k]);

                                } else {

                                    // If epsilon found, continue to next symbol

                                    if (productionSet[i][j + 1] != '\0') {

                                        j++;

                                    } else {

                                        // Add epsilon to Result set

                                        addToResultSet(Result, '$');

                                    }

                                }

                            }

                            break;

                        }

                    }

                }

            }

        }

    }

}

// Function to calculate FOLLOW set

void FOLLOW(char \*Result, char c)

{

    int i, j;

    // Adding "$" to the follow set of the start symbol

    if (productionSet[0][0] == c) {

        addToResultSet(Result, '$');

    }

    // Iterate over productions

    for (i = 0; i < numOfProductions; i++) {

        for (j = 2; productionSet[i][j] != '\0'; j++) {

            // If current character is found in production

            if (productionSet[i][j] == c) {

                // If there are symbols after the current one

                if (productionSet[i][j + 1] != '\0') {

                    // Calculate the first of the next Non-Terminal in the production

                    followfirst(productionSet[i][j + 1], i, (j + 1), Result);

                }

                // If current character is at the end of the production and it's not the start symbol

                if (productionSet[i][j + 1] == '\0' && c != productionSet[i][0]) {

                    // Calculate the follow of the Non-Terminal in the L.H.S. of the production

                    FOLLOW(Result, productionSet[i][0]);

                }

            }

        }

    }

}

// Function to calculate FIRST of a Non-Terminal symbol

void followfirst(char c, int i, int j, char Result[])

{

    int k;

    // If the symbol is a terminal, add it to the follow set

    if (!(isupper(c)))

        addToResultSet(Result, c);

    else {

        // Search for the FIRST set of the Non-Terminal symbol

        for (k = 0; k < numOfProductions; k++) {

            if (productionSet[k][0] == c)

                break;

        }

        // Include the FIRST set of the Non-Terminal symbol in the follow set

        while (productionSet[k][j] != '\0') {

            if (productionSet[k][j] != '#') {

                addToResultSet(Result, productionSet[k][j]);

                break;

            }

            // If '#' is found, move to the next symbol in the production

            if (productionSet[k][j] == '#') {

                if (productionSet[i][j + 1] != '\0')

                    followfirst(productionSet[i][j + 1], i, j + 1, Result);

                else if (productionSet[i][j + 1] == '\0' && c != productionSet[i][0])

                    FOLLOW(Result, productionSet[i][0]);

            }

            j++;

        }

    }

}

// Function to add a character to a result set

void addToResultSet(char Result[], char val)

{

    int k;

    for (k = 0; Result[k] != '\0'; k++)

        if (Result[k] == val)

            return;

    Result[k] = val;

    Result[k + 1] = '\0';

}

\

Output:

