COMPILER DESIGN LAB

LAB ASSIGNMENT – 4

NAME – ADITYARAJ SHRIVASTAVA

REG. NO. – 23BCE1968

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX 100
#define SIZE 100

char productions[SIZE][SIZE];
char nonTerminals[SIZE];
char firstSets[SIZE][SIZE];
char followSets[SIZE][SIZE];
int numProductions, numNonTerminals;

int isNonTerminal(char c)
{
    return isupper(c);
}

int isTerminal(char c)
{
    return (islower(c) || isdigit(c) || c == '+' || c == '-' || c == '*' || c == '/' || c == '(' || c == ')' || c == 'i');
}

int getNonTerminalIndex(char c)
{
    for (int i = 0; i < numNonTerminals; i++)
    {
        if (nonTerminals[i] == c)
            return i;
    }
    return -1;
}

void addToSet(char *set, char symbol)
{
    for (int i = 0; set[i] != '\0'; i++)
    {
        if (set[i] == symbol)
```

```
            return;
    }
    int len = strlen(set);
    set[len] = symbol;
    set[len + 1] = '\0';
}

void computeFirst()
{
    int changed;
    do
    {
        changed = 0;
        for (int i = 0; i < numProductions; i++)
        {
            char lhs = productions[i][0];
            char *rhs = productions[i] + 3; // Skip "X ->"

            int j = 0;
            while (rhs[j] != '\0' && rhs[j] != '|')
            {
                if (isTerminal(rhs[j]))
                {
                    if (strchr(firstSets[getNonTerminalIndex(lhs)], rhs[j]) == NULL)
                    {
                        addToSet(firstSets[getNonTerminalIndex(lhs)], rhs[j]);
                        changed = 1;
                    }
                    break;
                }
                else if (isNonTerminal(rhs[j]))
                {
                    int nonTermIndex = getNonTerminalIndex(rhs[j]);
                    for (int k = 0; firstSets[nonTermIndex][k] != '\0'; k++)
                    {
                        if (firstSets[nonTermIndex][k] != '#')
                        {
                            if (strchr(firstSets[getNonTerminalIndex(lhs)], firstSets[nonTermIndex][k]) == NULL)
                            {
                                addToSet(firstSets[getNonTerminalIndex(lhs)], firstSets[nonTermIndex][k]);
                                changed = 1;
                            }
                        }
                    }
                    if (strchr(firstSets[nonTermIndex], '#') == NULL)
                    {
                        break;
                    }
                }
                j++;
            }

            int allHaveEpsilon = 1;
            j = 0;
            while (rhs[j] != '\0' && rhs[j] != '|')
            {
                if (isTerminal(rhs[j]))
                {
                    allHaveEpsilon = 0;
                    break;
                }
                else if (isNonTerminal(rhs[j]))
                {
                    int idx = getNonTerminalIndex(rhs[j]);
                    if (strchr(firstSets[idx], '#') == NULL)
                    {
                        allHaveEpsilon = 0;
                        break;
                    }
                }
                j++;
            }
            if (allHaveEpsilon && strchr(firstSets[getNonTerminalIndex(lhs)], '#') == NULL)
            {
                addToSet(firstSets[getNonTerminalIndex(lhs)], '#');
                changed = 1;
            }

            while (rhs[j] != '\0')
            {
                if (rhs[j] == '|')
                {
                    j++;
                    int k = j;
                    while (rhs[k] != '\0' && rhs[k] != '|')
                    {
```

```
                        if (isTerminal(rhs[k]))
                        {
                            if (strchr(firstSets[getNonTerminalIndex(lhs)], rhs[k]) == NULL)
                            {
                                addToSet(firstSets[getNonTerminalIndex(lhs)], rhs[k]);
                                changed = 1;
                            }
                            goto next_prod;
                        }
                        else if (isNonTerminal(rhs[k]))
                        {
                            int idx = getNonTerminalIndex(rhs[k]);
                            for (int l = 0; firstSets[idx][l] != '\0'; l++)
                            {
                                if (firstSets[idx][l] != '#')
                                {
                                    if (strchr(firstSets[getNonTerminalIndex(lhs)], firstSets[idx][l]) == NULL)
                                    {
                                        addToSet(firstSets[getNonTerminalIndex(lhs)], firstSets[idx][l]);
                                        changed = 1;
                                    }
                                }
                            }
                            if (strchr(firstSets[idx], '#') == NULL)
                            {
                                goto next_prod;
                            }
                        }
                        k++;
                    }

                    int altAllNull = 1;
                    k = j;
                    while (rhs[k] != '\0' && rhs[k] != '|')
                    {
                        if (isTerminal(rhs[k]) || (isNonTerminal(rhs[k]) && strchr(firstSets[getNonTerminalIndex(rhs[k])], '#') ==
NULL))
                        {
                            altAllNull = 0;
                            break;
                        }
                        k++;
                    }
                    if (altAllNull && strchr(firstSets[getNonTerminalIndex(lhs)], '#') == NULL)
                    {
                        addToSet(firstSets[getNonTerminalIndex(lhs)], '#');
                        changed = 1;
                    }
                }
                j++;
            }
        next_prod:;
        }
    } while (changed);
}
void computeFollow()
{
    addToSet(followSets[0], '$');

    int changed;
    do
    {
        changed = 0;
        for (int i = 0; i < numProductions; i++)
        {
            char lhs = productions[i][0];
            char *rhs = productions[i] + 3;

            for (int j = 0; rhs[j] != '\0'; j++)
            {
                if (isNonTerminal(rhs[j]))
                {
                    int current = getNonTerminalIndex(rhs[j]);
                    int next = j + 1;

                    if (rhs[next] == '|')
                    {
                        // Skip to after |
                        while (rhs[next] != '\0' && rhs[next] != '|')
                            next++;
                        next++;
                    }

                    if (rhs[next] != '\0' && rhs[next] != '|')
                    {
                        if (isTerminal(rhs[next]))
```

```
                    {
                        if (strchr(followSets[current], rhs[next]) == NULL)
                        {
                            addToSet(followSets[current], rhs[next]);
                            changed = 1;
                        }
                    }
                    else if (isNonTerminal(rhs[next]))
                    {
                        int nextNT = getNonTerminalIndex(rhs[next]);
                        for (int k = 0; firstSets[nextNT][k] != '\0'; k++)
                        {
                            if (firstSets[nextNT][k] != '#')
                            {
                                if (strchr(followSets[current], firstSets[nextNT][k]) == NULL)
                                {
                                    addToSet(followSets[current], firstSets[nextNT][k]);
                                    changed = 1;
                                }
                            }
                        }

                        if (strchr(firstSets[nextNT], '#') != NULL)
                        {
                            for (int k = 0; followSets[getNonTerminalIndex(lhs)][k] != '\0'; k++)
                            {
                                if (strchr(followSets[current], followSets[getNonTerminalIndex(lhs)][k]) == NULL)
                                {
                                    addToSet(followSets[current], followSets[getNonTerminalIndex(lhs)][k]);
                                    changed = 1;
                                }
                            }
                        }
                    }
                }
                else
                {
                    // End of RHS, add Follow of LHS
                    for (int k = 0; followSets[getNonTerminalIndex(lhs)][k] != '\0'; k++)
                    {
                        if (strchr(followSets[current], followSets[getNonTerminalIndex(lhs)][k]) == NULL)
                        {
                            addToSet(followSets[current], followSets[getNonTerminalIndex(lhs)][k]);
                            changed = 1;
                        }
                    }
                }
            }
        }
    } while (changed);
}
int main()
{
    printf("Enter number of productions: ");
    scanf("%d", &numProductions);
    getchar();

    printf("Enter productions ->  \n");
    for (int i = 0; i < numProductions; i++)
    {
        printf("Production %d: ", i + 1);
        fgets(productions[i], SIZE, stdin);
        productions[i][strcspn(productions[i], "\n")] = 0;

        char lhs = productions[i][0];
        int found = 0;
        for (int j = 0; j < numNonTerminals; j++)
        {
            if (nonTerminals[j] == lhs)
            {
                found = 1;
                break;
            }
        }
        if (!found)
        {
            nonTerminals[numNonTerminals++] = lhs;
        }
    }
    for (int i = 0; i < numNonTerminals; i++)
    {
        firstSets[i][0] = '\0';
        followSets[i][0] = '\0';
    }
    computeFirst();
```

```
    computeFollow();
    printf("\nFirst Sets:\n");
    for (int i = 0; i < numNonTerminals; i++)
    {
        printf("First(%c) = { ", nonTerminals[i]);
        for (int j = 0; firstSets[i][j] != '\0'; j++)
        {
            printf("%c ", firstSets[i][j]);
        }       printf("}\n");
    }
    printf("\nFollow Sets:\n");
    for (int i = 0; i < numNonTerminals; i++)
    {
        printf("Follow(%c) = { ", nonTerminals[i]);
        for (int j = 0; followSets[i][j] != '\0'; j++)
        {
            printf("%c ", followSets[i][j]);
        }
        printf("}\n");
    }
    return 0;
}
```

Output: