LAB-9

Aim:

Objective (1): Construct an AST for the given expression: a + a * (b-c) + (b-c) * d

Objective (2): Implement S-R Parsing:

E --> E + E

E --> E * E

E --> (E)

E --> id

Code 1:

```
// write a program to Construct an AST for expression a+a*(b-c)+(b-c)*d

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Define the structure for an AST Node
typedef struct ASTNode {
    char data[10];           // Value of the node (operator or operand)
    struct ASTNode *left;    // Left child
    struct ASTNode *right;   // Right child
} ASTNode;

// Function to create a new AST node
ASTNode* createNode(char *data) {
    ASTNode *newNode = (ASTNode *)malloc(sizeof(ASTNode));
    strcpy(newNode->data, data);
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Function to display the AST in preorder traversal
void printAST(ASTNode *root, int level) {
    if (root == NULL)
        return;

    // Indent to visualize the tree structure
    for (int i = 0; i < level; i++)
        printf("    ");
    printf("%s\n", root->data);

    printAST(root->left, level + 1);
    printAST(root->right, level + 1);
}

int main() {
    // Create nodes for the expression a + a * (b - c) + (b - c) * d
    ASTNode *root = createNode("+");

    // Left subtree of root: a + a * (b - c)
    root->left = createNode("+");
    root->left->left = createNode("a");
    root->left->right = createNode("*");
```

```
    root->left->right->left = createNode("a");
    root->left->right->right = createNode("-");
    root->left->right->right->left = createNode("b");
    root->left->right->right->right = createNode("c");

    // Right subtree of root: (b - c) * d
    root->right = createNode("*");
    root->right->left = createNode("-");
    root->right->left->left = createNode("b");
    root->right->left->right = createNode("c");
    root->right->right = createNode("d");

    // Print the AST
    printf("Abstract Syntax Tree (AST):\n");
    printAST(root, 0);

    return 0;
}
```

Output:

```
[chiragsharma@192 Lab 9 % ./task1
 Abstract Syntax Tree (AST):
 +
     +
         a
         *
             a
             -
                 b
                 c
     *
         -
             b
             c
         d
```

Code 2:

```
// Implement SR Parsing

// E->E+E

// E->E*E

// E->(E)

// E->id


#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// Global Variables

int z = 0, i = 0, j = 0, c = 0;
```

```
char a[20], ac[20], stk[20], act[10];

// Function to check for reductions
void check() {
    strcpy(ac, "REDUCE TO E -> ");

    // Checking for reduction rules
    for (z = 0; z < c; z++) {
        // Checking for E -> id
        if (stk[z] == 'i' && stk[z + 1] == 'd') {
            printf("%sid\n", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
            break;
        }
    }

    for (z = 0; z < c - 2; z++) {
        // Checking for E -> E + E
        if (stk[z] == 'E' && stk[z + 1] == '+' && stk[z + 2] == 'E') {
            printf("%sE+E\n", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            stk[z + 2] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
            i -= 2;
            break;
        }

        // Checking for E -> E * E
        if (stk[z] == 'E' && stk[z + 1] == '*' && stk[z + 2] == 'E') {
            printf("%sE*E\n", ac);
            stk[z] = 'E';
```

```
            stk[z + 1] = '\0';
            stk[z + 2] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
            i -= 2;
            break;
        }
    }

    for (z = 0; z < c - 2; z++) {
        // Checking for E -> ( E )
        if (stk[z] == '(' && stk[z + 1] == 'E' && stk[z + 2] == ')') {
            printf("%s(E)\n", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            stk[z + 2] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
            i -= 2;
            break;
        }
    }
    return;
}

// Main Function
int main() {
    printf("GRAMMAR is -\nE -> E + E\nE -> E * E\nE -> ( E )\nE -> id\n");

    strcpy(a, "id+id*id");
    c = strlen(a);

    strcpy(act, "SHIFT");

    printf("\nstack \t input \t action\n");

    printf("\n$\t%s$\t", a);
```

```
    // Parsing loop
    for (i = 0; j < c; i++, j++) {
        // Perform SHIFT operation
        printf("%s\n", act);
        stk[i] = a[j];
        stk[i + 1] = '\0';
        a[j] = ' ';
        printf("$%s\t%s$\t", stk, a);

        // Check for reductions
        check();
    }

    check();

    if (stk[0] == 'E' && stk[1] == '\0') {
        printf("ACCEPT\n");
    } else {
        printf("REJECT\n");
    }
}
```

Output:

```
GRAMMAR is -
E -> E + E
E -> E * E
E -> ( E )
E -> id

stack    input   action

$       id+id*id$       SHIFT
$i       d+id*id$       SHIFT
$id       +id*id$       REDUCE TO E -> id

$E        +id*id$       SHIFT
$E         id*id$       SHIFT
$E          d*id$       SHIFT
$E           *id$       REDUCE TO E -> id

$E           *id$       SHIFT
$E            id$       SHIFT
$E             d$       SHIFT
$E              $       REDUCE TO E -> id

$E              $       ACCEPT
```