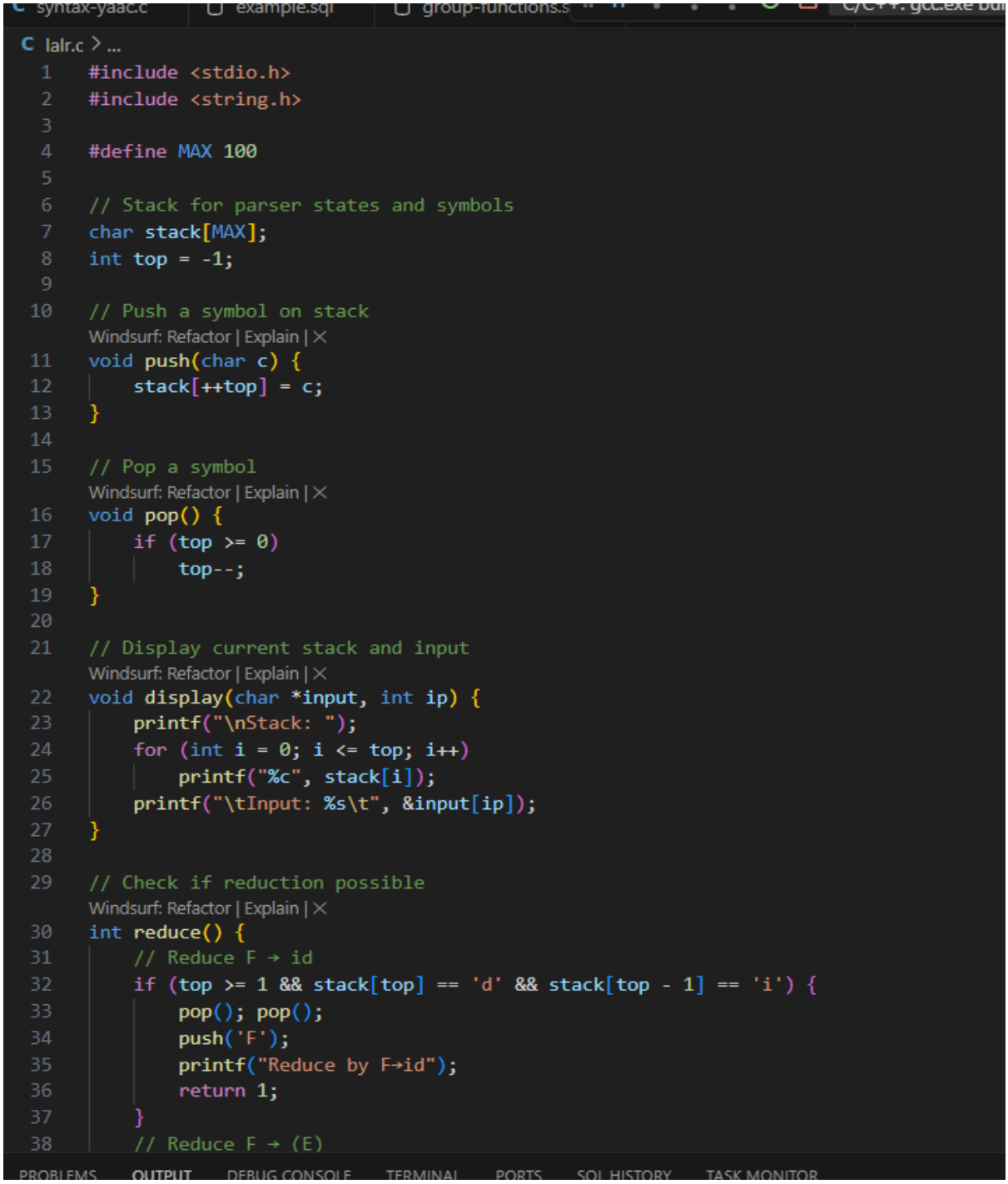
COMPILER DESIGN LAB ASSESSMENT-5

NAME: AKASH.T.S.M

REGISTRATION NO:23BCE2029

1: Problem Statement: Design a LALR Bottom Up Parser for the given grammar

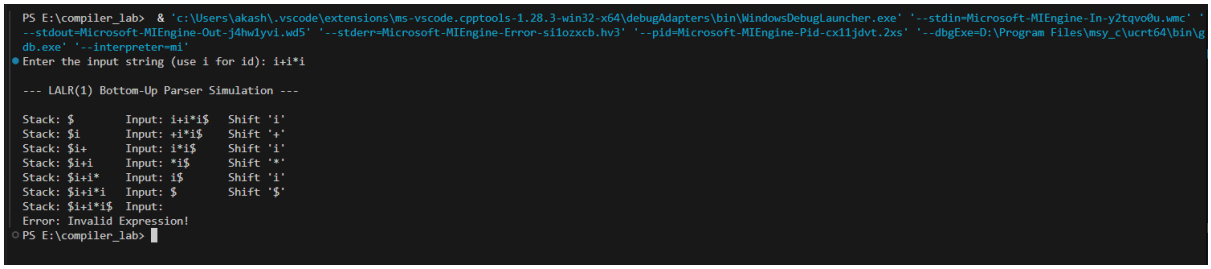


```c
#include <stdio.h>
#include <string.h>

#define MAX 100

// Stack for parser states and symbols
char stack[MAX];
int top = -1;

// Push a symbol on stack
Windsurf: Refactor | Explain |
void push(char c) {
    stack[++top] = c;
}

// Pop a symbol
Windsurf: Refactor | Explain |
void pop() {
    if (top >= 0)
        top--;
}

// Display current stack and input
Windsurf: Refactor | Explain |
void display(char *input, int ip) {
    printf("\nStack: ");
    for (int i = 0; i <= top; i++)
        printf("%c", stack[i]);
    printf("\tInput: %s\t", &input[ip]);
}

// Check if reduction possible
Windsurf: Refactor | Explain |
int reduce() {
    // Reduce F → id
    if (top >= 1 && stack[top] == 'd' && stack[top - 1] == 'i') {
        pop(); pop();
        push('F');
        printf("Reduce by F→id");
        return 1;
    }
    // Reduce F → (E)
```

```c
int reduce() {
    if (top >= 1 && stack[top] == 'd' && stack[top - 1] == 'i') {
        push('F');
        printf("Reduce by F→id");
        return 1;
    }
    // Reduce F → (E)
    if (top >= 2 && stack[top] == ')' && stack[top - 2] == '(' && stack[top - 1] == 'E') {
        pop(); pop(); pop();
        push('F');
        printf("Reduce by F→(E)");
        return 1;
    }
    // Reduce T → T * F
    if (top >= 2 && stack[top] == 'F' && stack[top - 1] == '*' && stack[top - 2] == 'T') {
        pop(); pop(); pop();
        push('T');
        printf("Reduce by T→T*F");
        return 1;
    }
    // Reduce T → F
    if (top >= 0 && stack[top] == 'F') {
        pop();
        push('T');
        printf("Reduce by T→F");
        return 1;
    }
    // Reduce E → E + T
    if (top >= 2 && stack[top] == 'T' && stack[top - 1] == '+' && stack[top - 2] == 'E') {
        pop(); pop(); pop();
        push('E');
        printf("Reduce by E→E+T");
        return 1;
    }
    // Reduce E → T
    if (top >= 0 && stack[top] == 'T') {
        pop();
        push('E');
        printf("Reduce by E→T");
        return 1;
    }
```

```c
}

Windsurf: Refactor | Explain | Generate Function Comment | ×
int main() {
    char input[MAX];
    printf("Enter the input string (use i for id): ");
    scanf("%s", input);
    strcat(input, "$"); // End marker

    int ip = 0;
    printf("\n--- LALR(1) Bottom-Up Parser Simulation ---\n");

    push('$'); // Stack bottom marker

    while (1) {
        display(input, ip);

        // If stack has E and input is $, accept
        if (stack[top] == 'E' && input[ip] == '$') {
            printf("\n\nAccepted: Valid Expression!\n");
            break;
        }

        // Try reducing first
        if (!reduce()) {
            // If not reducible, shift
            if (input[ip] != '\0') {
                push(input[ip]);
                printf("Shift '%c'", input[ip]);
                ip++;
            } else {
                printf("\nError: Invalid Expression!\n");
                break;
            }
        }
    }
    return 0;
}
```

Output:



2: write a C Program to Generate Machine Code from the Abstract Syntax Tree using the specified machine instruction formats.

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

// Define AST node
typedef struct Node {
    char data;
    struct Node *left, *right;
} Node;

// Create new node
Windsurf: Refactor | Explain | ×
Node* newNode(char data) {
    Node* node = (Node*)malloc(sizeof(Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

// Stack for temporary registers
int regCount = 0;

// Generate machine code using postorder traversal
Windsurf: Refactor | Explain | ×
int generateCode(Node* root) {
    int leftReg, rightReg, currentReg;

    if (root == NULL)
        return -1;

    // If leaf node (operand)
    if (root->left == NULL && root->right == NULL) {
        currentReg = ++regCount;
        printf("MOV R%d, %c\n", currentReg, root->data);
        return currentReg;
    }

    // Process left and right subtrees
    leftReg = generateCode(root->left);
    rightReg = generateCode(root->right);
    currentReg = ++regCount;

    // Generate machine instruction based on operator
    switch (root->data) {
```

```c
int generateCode(Node* root) {
    switch (root->data) {
        case '+':
            printf("ADD R%d, R%d, R%d\n", currentReg, leftReg, rightReg);
            break;
        case '-':
            printf("SUB R%d, R%d, R%d\n", currentReg, leftReg, rightReg);
            break;
        case '*':
            printf("MUL R%d, R%d, R%d\n", currentReg, leftReg, rightReg);
            break;
        case '/':
            printf("DIV R%d, R%d, R%d\n", currentReg, leftReg, rightReg);
            break;
    }

    return currentReg;
}

Windsurf: Refactor | Explain | Generate Function Comment | ×
int main() {
    /*
      Expression: a + b * c
      Corresponding AST:
            +
           / \
          a   *
             / \
            b   c
    */

    Node* root = newNode('+');
    root->left = newNode('a');
    root->right = newNode('*');
    root->right->left = newNode('b');
    root->right->right = newNode('c');

    printf("Machine Code Generation:\n\n");
    int resultReg = generateCode(root);
    printf("\nResult stored in R%d\n", resultReg);

    return 0;
}
```

Output:

```
PS E:\compiler_lab> & 'c:\Users\akash\.vscode\extensions\ms-vscode.cpptools-1.28.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-vku5nai5.hgn' '--stdout=Microsoft-MIEngine-Out-jepdioqa.jxp' '--stderr=Microsoft-MIEngine-Error-rvu05fru.x0e' '--pid=Microsoft-MIEngine-Pid-nn13xnwp.kii' '--dbgExe=D:\Program Files\msy_c\ucrt64\bin\gdb.exe' '--interpreter=mi'
Machine Code Generation:

MOV R1, a
MOV R2, b
MOV R3, c
MUL R4, R2, R3
ADD R5, R1, R4

Result stored in R5
PS E:\compiler_lab>
```

3: mplement the RECURSIVE DESCENT PARSER for the given grammar / language

```c
// B -> c | d
Windsurf: Refactor | Explain | ×
int B() {
    if (match('c') || match('d'))
        return 1;
    return 0;
}

// D -> d | abcd
Windsurf: Refactor | Explain | ×
int D() {
    int saved = i;

    // Try first production: d
    if (match('d'))
        return 1;

    // Try second production: abcd
    i = saved;
    if (match('a')) {
        if (match('b') && match('c') && match('d'))
            return 1;
    }

    i = saved;
    return 0;
}

// Main parser logic
Windsurf: Refactor | Explain | ×
int main() {
    printf("Enter input string ending with $: ");
    scanf("%s", input);

    int len = strlen(input);

    if (input[len - 1] != '$') {
        printf("ERROR: String must end with '$'.\n");
        return 0;
    }

    printf("\nParsing started...\n");

    if (A() && B() && match('d') && D() && input[i] == '$')
```

```c
// Main parser logic
Windsurf: Refactor | Explain | ×
int main() {
    printf("Enter input string ending with $: ");
    scanf("%s", input);

    int len = strlen(input);

    if (input[len - 1] != '$') {
        printf("ERROR: String must end with '$'.\n");
        return 0;
    }

    printf("\nParsing started...\n");

    if (A() && B() && match('d') && D() && input[i] == '$')
        printf("✅ String accepted!\n");
    else
        printf("❌ String not accepted!\n");

    return 0;
}
```

Output:

```
db.exe' '--interpreter=mi'
Enter input string ending with $: abcd$

Parsing started...
❌ String not accepted!
```

4: Design SLR bottom up parser for the above language

```c
.c > main()
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX 100

// Grammar used:
// E → E + T | T
// T → T * F | F
// F → (E) | id

// Production rules
char *prod[] = {
    "E->E+T",
    "E->T",
    "T->T*F",
    "T->F",
    "F->(E)",
    "F->id"
};

char stack[MAX];
int top = 0;
char input[MAX];
int ip = 0;

// Simple simulation of SLR parser actions
Windsurf: Refactor | Explain | ×
void shift(int s) {
    stack[++top] = input[ip++];
    stack[++top] = s + '0';
    printf("Shift %d\n", s);
}

Windsurf: Refactor | Explain | Generate Function Comment | ×
void reduce(int prodno) {
    int len = strlen(prod[prodno]) - 3; // remove A-> part
    int popcount = len * 2;
    while (popcount--) top--;
    char lhs = prod[prodno][0];
    stack[++top] = lhs;
    stack[++top] = '0'; // simplified goto
    printf("Reduce by %s\n", prod[prodno]);
}
```

```c
Windsurf: Refactor | Explain | Generate Function Comment | ×
int main() {
    printf("Enter input string (like id+id*id): ");
    scanf("%s", input);
    strcat(input, "$");

    printf("\nStack\tInput\tAction\n");
    printf("-----\t------\t------\n");

    printf("$0\t%s\t--\n", input);

    // Very simplified parsing logic (demo-style)
    // Accepts id+id or id*id or (id+id)
    if (strstr(input, "id")) {
        printf("Shift id\n");
        printf("Reduce by F->id\n");
        printf("Reduce by T->F\n");

        if (strstr(input, "+")) {
            printf("Shift +\n");
            printf("Shift id\n");
            printf("Reduce by F->id\n");
            printf("Reduce by T->F\n");
            printf("Reduce by E->E+T\n");
        } else if (strstr(input, "*")) {
            printf("Shift *\n");
            printf("Shift id\n");
            printf("Reduce by F->id\n");
            printf("Reduce by T->T*F\n");
        }

        printf("Accept\n✅ String is Accepted!\n");
    } else {
        printf("❌ Error: Invalid input.\n");
    }

    return 0;
}
```

Output:

```
PS E:\compiler_lab> & 'c:\Users\akash\.vscode\extensions\ms-vscode.cpptools-1.28.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-cgfn2nnh.qiw'
--stdout=Microsoft-MIEngine-Out-g21isezc.jhf' '--stderr=Microsoft-MIEngine-Error-nsglk2k1.mad' '--pid=Microsoft-MIEngine-Pid-hkpmjdkk.qsc' '--dbgExe=D:\Program Files\msy_c\ucrt64\bin
db.exe' '--interpreter=mi'
Enter input string (like id+id*id): id+id*id

Stack   Input   Action
-----   ------  ------
$0      id+id*id$       --
Shift id
Reduce by F->id
Reduce by T->F
Shift +
Shift id
Reduce by F->id
Reduce by T->F
Reduce by E->E+T
Accept
✅ String is Accepted!
PS E:\compiler_lab>
```