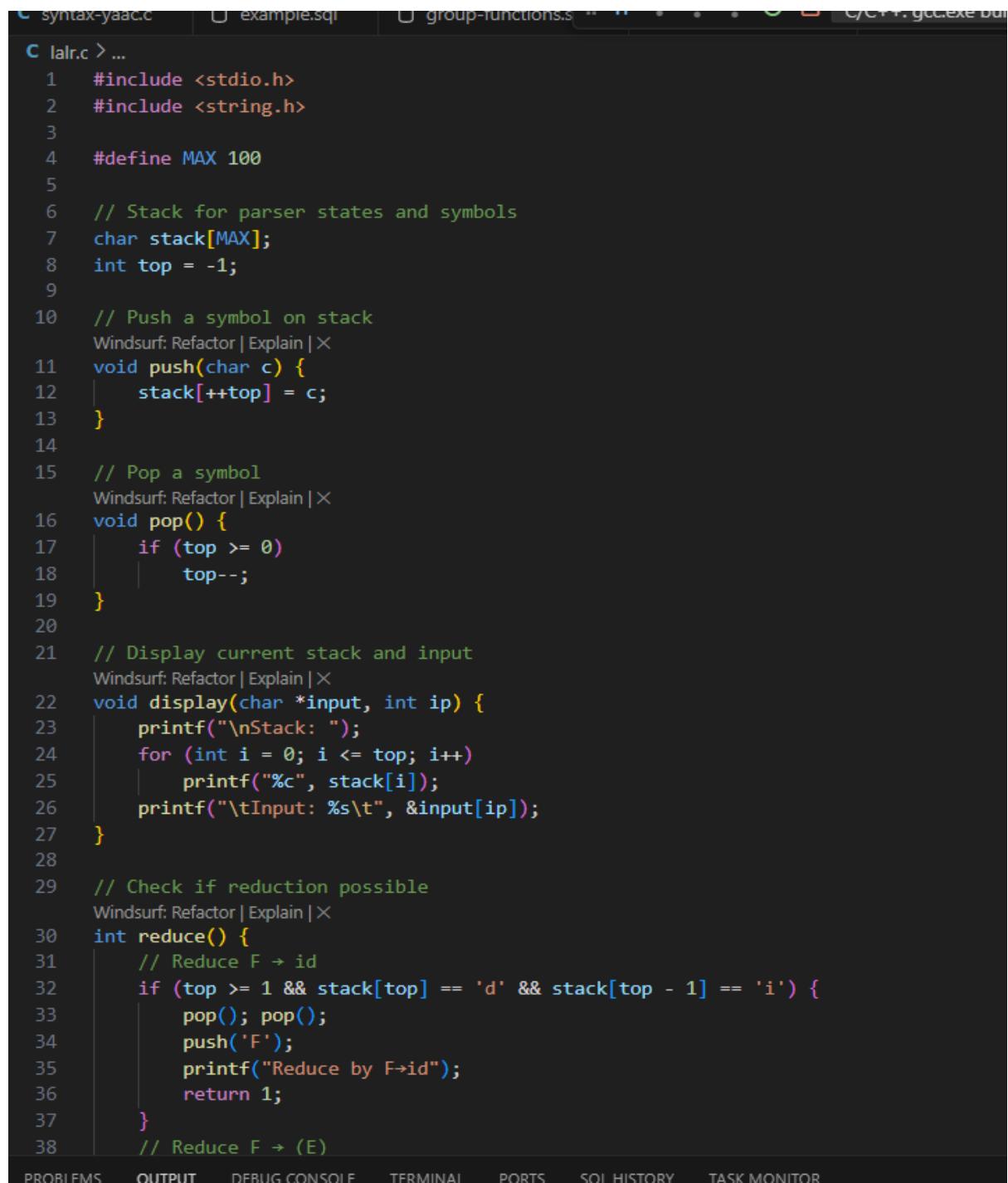


COMPILER DESIGN LAB ASSESSMENT-5

NAME: AKASH.T.S.M

REGISTRATION NO:23BCE2029

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



The screenshot shows a code editor window with the file 'lalr.c' open. The code implements a bottom-up parser using a stack. It includes functions for pushing and popping symbols from the stack, displaying the current stack and input, and checking if a reduction is possible. The code is annotated with WindSurf refactoring tools.

```
C syntax-yaac.c example.sql group-functions.s C/C++ g++ execute build
C lalr.c > ...
1 #include <stdio.h>
2 #include <string.h>
3
4 #define MAX 100
5
6 // Stack for parser states and symbols
7 char stack[MAX];
8 int top = -1;
9
10 // Push a symbol on stack
11 void push(char c) {
12     stack[++top] = c;
13 }
14
15 // Pop a symbol
16 void pop() {
17     if (top >= 0)
18         top--;
19 }
20
21 // Display current stack and input
22 void display(char *input, int ip) {
23     printf("\nStack: ");
24     for (int i = 0; i <= top; i++)
25         printf("%c", stack[i]);
26     printf("\tInput: %s\t", &input[ip]);
27 }
28
29 // Check if reduction possible
30 int reduce() {
31     // Reduce F → id
32     if (top >= 1 && stack[top] == 'd' && stack[top - 1] == 'i') {
33         pop(); pop();
34         push('F');
35         printf("Reduce by F->id");
36         return 1;
37     }
38     // Reduce F → (E)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL HISTORY TASK MONITOR

```

30     int reduce() {
31         if (top >= 1 && stack[top] == 'd' && stack[top - 1] == 'i') {
32             push('F');
33             printf("Reduce by F->i");
34             return 1;
35         }
36         // Reduce F -> (E)
37         if (top >= 2 && stack[top] == ')' && stack[top - 2] == '(' && stack[top - 1] == 'E') {
38             pop(); pop(); pop();
39             push('F');
40             printf("Reduce by F->(E)");
41             return 1;
42         }
43         // Reduce T -> T * F
44         if (top >= 2 && stack[top] == 'F' && stack[top - 1] == '*' && stack[top - 2] == 'T') {
45             pop(); pop(); pop();
46             push('T');
47             printf("Reduce by T->T*F");
48             return 1;
49         }
50         // Reduce T -> F
51         if (top >= 0 && stack[top] == 'F') {
52             pop();
53             push('T');
54             printf("Reduce by T->F");
55             return 1;
56         }
57         // Reduce E -> E + T
58         if (top >= 2 && stack[top] == 'T' && stack[top - 1] == '+' && stack[top - 2] == 'E') {
59             pop(); pop(); pop();
60             push('E');
61             printf("Reduce by E->E+T");
62             return 1;
63         }
64         // Reduce E -> T
65         if (top >= 0 && stack[top] == 'T') {
66             pop();
67             push('E');
68             printf("Reduce by E->T");
69             return 1;
70         }
71     }
72 }
```

```

}

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("%c", input[ip]);
                ip++;
            } else {
                printf("\nError: Invalid Expression!\n");
                break;
            }
        }
    }
    return 0;
}
```

Output:

```
PS E:\compiler_lab> & 'c:\Users\akash\vscode\extensions\ms-vscode.cppTools-1.28.3-win32-x64\debug\adapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-y2tqvobu.wmc' '--stdout=Microsoft-MIEngine-Out-j4hwiyvi.wd5' '--stderr=Microsoft-MIEngine-Error-si1lozxcb.hv3' '--pid=Microsoft-MIEngine-Pid-cx11jdvt.2xs' '--dbgExe=D:\Program Files\msy_c\ucrt64\bgdb.exe' '--interpreter=i'
● Enter the input string (use i for id): i+i*
-- LALR(1) Bottom-Up Parser Simulation --
Stack: $      Input: i+i*$   Shift 'i'
Stack: $i     Input: +i*i$   Shift '+'
Stack: $i+    Input: i*i$   Shift 'i'
Stack: $i+i   Input: *i$   Shift '*'
Stack: $i+i*  Input: i$   Shift 'i'
Stack: $i+i*i Input: $   Shift '$'
Stack: $i+i*i$ Input:
Error: Invalid Expression!
○ PS E:\compiler_lab>
```

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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 // Define AST node
6 typedef struct Node {
7     char data;
8     struct Node *left, *right;
9 } Node;
10
11 // Create new node
12 Node* newNode(char data) {
13     Node* node = (Node*)malloc(sizeof(Node));
14     node->data = data;
15     node->left = node->right = NULL;
16     return node;
17 }
18
19 // Stack for temporary registers
20 int regCount = 0;
21
22 // Generate machine code using postorder traversal
23 int generateCode(Node* root) {
24     int leftReg, rightReg, currentReg;
25
26     if (root == NULL)
27         return -1;
28
29     // If leaf node (operand)
30     if (root->left == NULL && root->right == NULL) {
31         currentReg = ++regCount;
32         printf("MOV R%d, %c\n", currentReg, root->data);
33         return currentReg;
34     }
35
36     // Process left and right subtrees
37     leftReg = generateCode(root->left);
38     rightReg = generateCode(root->right);
39     currentReg = ++regCount;
40
41     // Generate machine instruction based on operator
42     switch (root->data) {
```

```

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 | X
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.cpp-tools-1.28.3-win32-x64\debugAdapters\bIn\WindowsDebugLauncher.exe" '--stdin=Microsoft-MIEngine-In-vku5na15.hgn' '--stdout=Microsoft-MIEngine-Out-jepdjq0.jxp' '--stderr=Microsoft-MIEngine-Error-rvu05fru.x0e' '--pid=Microsoft-MIEngine-Pid-nn13xnwp.kii' '--dbgExe=D:\Program Files\msy_c\ucrt64\bin\gdb.exe' '-interpreter=m1'
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: Implement the RECURSIVE DESCENT PARSER for the given grammar / language

```
syntax-yaac.c | example.sql | group-functions.sql | operators-sql.sql
C recursive-parser.c > ...
1 #include <stdio.h>
2 #include <string.h>
3
4 char input[20]; // input string
5 int i = 0; // pointer to current input symbol
6
7 // Function declarations
8 int A();
9 int B();
10 int D();
11
12 // Match a terminal symbol
Windsurf: Refactor | Explain | ×
13 int match(char c) {
14     if (input[i] == c) {
15         i++;
16         return 1;
17     }
18     return 0;
19 }
20
21 // A -> bc | ab
Windsurf: Refactor | Explain | ×
22 int A() {
23     int saved = i; // save index in case first rule fails
24
25     // Try first production: bc
26     if (match('b')) {
27         if (match('c'))
28             return 1;
29         else {
30             i = saved; // backtrack
31         }
32     }
33
34     // Try second production: ab
35     i = saved;
36     if (match('a')) {
37         if (match('b'))
38             return 1;
39     }
40
41     i = saved;
42     return 0;
43 }
```

```

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

// D -> d | abcd
Windsurf: Refactor | Explain | X
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 | X
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] == '$')

```

```

// Main parser logic
Windsurf: Refactor | Explain | X
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 /* 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 | X
void shift(int s) {
    stack[++top] = input[ip++];
    stack[++top] = s + '0';
    printf("Shift %d\n", s);
}

Windsurf: Refactor | Explain | Generate Function Comment | X
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]);
}
```

```

Windsurf: Refactor | Explain | Generate Function Comment | X
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.cppptools-1.28.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-cgfn2nnh.qiw'
--stdout=Microsoft-MIEngine-Out-g2lisezc.jhf' '--stderr=Microsoft-MIEngine-Error-nsglk2kl.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> []

```

