COMPILER DESIGN LAB – 6

ADITYA AGARWAL **220905106** **ROLL NO. 14**

Q1 - Write a recursive descent parser for the following simple grammars.

GRAMMAR -

S → a | > | (T)
T → T, S|S

Left Recursion: T → T, S | S

After left recursion removal:

S → a | > | (T)

T → S T'

T' → , S T' | ϵ

First Sets:

- First(S) = {a, >, (}
- First(T) = {a, >, (}
- First(T') = {, , ϵ}

Follow Sets:

- Follow(S) = {, ,), $}
- Follow(T) = {), $}
- Follow(T') = {, ,), $}

CODE:

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

int curr = 0;
char str[100];

// Function declarations
void S();
void T();
void invalid();
void valid();
void match(char expected);
void skipSpaces();

// Error handling
void invalid() {
```

```
    printf("-----------------ERROR!----------------\n");
    exit(0);
}

void valid() {
    printf("----------------SUCCESS!---------------\n");
    exit(0);
}

// Utility functions
void skipSpaces() {
    while (str[curr] == ' ' || str[curr] == '\t') {
        curr++;
    }
}

void match(char expected) {
    skipSpaces();
    if (str[curr] == expected) {
        curr++;
    } else {
        printf("Expected '%c' but found '%c'\n", expected, str[curr]);
        invalid();
    }
}

// Grammar rules
void S() {
    // S → a | > | ( T )
    skipSpaces();

    switch(str[curr]) {
        case 'a':
            match('a');
            break;
        case '>':
            match('>');
            break;
        case '(':
            match('(');
            T();
            match(')');
            break;
        default:
            printf("Invalid symbol in S: '%c'\n", str[curr]);
            invalid();
    }
}

void T() {
    // T → T, S | S
    S();  // First parse S
```

```
    skipSpaces();
    while (str[curr] == ',') {
        match(',');
        skipSpaces();
        S();
        skipSpaces();
    }
}

int main() {
    printf("Enter String: ");
    fgets(str, 100, stdin);

    // Remove newline if present
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == '\n') {
            str[i] = '\0';
            break;
        }
    }

    S();  // Start parsing with S
    skipSpaces();

    if (str[curr] == '\0') {
        valid();
    } else {
        printf("Unexpected characters after valid expression: '%s'\n", &str[curr]);
        invalid();
    }

    return 0;
}
```

OUTPUT:

```
Enter String: a
----------SUCCESS!----------

Enter String: (a
Expected ')' but found "
----------ERROR!----------
```

Q2 - Write a recursive descent parser for the following simple grammars.

GRAMMAR -

S → UVW
U → (S) | aSb | d
V → aV | ϵ
W → cW | ϵ

Left Recursion: V → aV | ϵ AND W → cW | ϵ

After left recursion removal:

S → UVW

U → (S) | aSb | d

V → ϵ V'

V' → a V' | ϵ

W → ϵ W'

W' → c W' | ϵ

First Sets:

- First(S) = { (, a, d }
- First(U) = { (, a, d }
- First(V) = { a, ϵ }
- First(W) = { c, ϵ }

Follow Sets:

- Follow(S) = {), $ }
- Follow(U) = {), $ }
- Follow(V) = {), $ }
- Follow(W) = {), $ }

CODE:

```
#include <stdio.h>

#include <stdlib.h>


int curr = 0;

char str[100];


// Function declarations

void S();

void U();

void V();
```

```
void W();
void invalid();
void valid();
void match(char expected);
void skipSpaces();

// Error handling
void invalid() {
    printf("-----------------ERROR!----------------\n");
    exit(0);
}

void valid() {
    printf("----------------SUCCESS!---------------\n");
    exit(0);
}

// Utility functions
void skipSpaces() {
    while (str[curr] == ' ' || str[curr] == '\t') {
        curr++;
    }
}

void match(char expected) {
    skipSpaces();
    if (str[curr] == expected) {
        curr++;
    } else {
        printf("Expected '%c' but found '%c'\n", expected, str[curr]);
        invalid();
    }
```

```
}

// Grammar rules

// S → UVW
void S() {
    skipSpaces();
    U();  // Parse U
    V();  // Parse V
    W();  // Parse W
}

// U → (S) | aSb | d
void U() {
    skipSpaces();

    switch(str[curr]) {
        case '(':
            match('(');
            S();
            match(')');
            break;

        case 'a':
            match('a');
            S();
            match('b');
            break;

        case 'd':
            match('d');
            break;
```

```
        default:
            printf("Invalid symbol in U: '%c'\n", str[curr]);
            invalid();
    }
}

// V → aV |
void V() {
    skipSpaces();

    // Check if we have 'a', otherwise assume epsilon
    while (str[curr] == 'a') {
        match('a');
    }
    // epsilon production requires no action
}

// W → cW |
void W() {
    skipSpaces();

    // Check if we have 'c', otherwise assume epsilon
    while (str[curr] == 'c') {
        match('c');
    }
    // epsilon production requires no action
}

int main() {
    printf("Enter String: ");
    fgets(str, 100, stdin);
```

```
    // Remove newline if present
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == '\n') {
            str[i] = '\0';
            break;
        }
    }

    S();  // Start parsing with S
    skipSpaces();

    if (str[curr] == '\0') {
        valid();
    } else {
        printf("Unexpected characters after valid expression: '%s'\n", &str[curr]);
        invalid();
    }

    return 0;
}
```

OUTPUT:

```
Enter String: d
----------------SUCCESS!---------------
Enter String: (a)
Invalid symbol in U: ')'
----------------ERROR!----------------
```

Q3 - Write a recursive descent parser for the following simple grammars.

GRAMMAR -

S → aAcBe

A → Ab|b

B → d

Left Recursion: A → Ab | b

After left recursion removal:

S → aAcBe

A → bA'

A' → bA' | ϵ

B → d

First Sets:

- First(S) = { a }
- First(A) = { b }
- First(A') = { b, ϵ }
- First(B) = { d }

Follow Sets:

- Follow(S) = { $ }
- Follow(A) = { c }
- Follow(A') = { c, $ }
- Follow(B) = { e }

CODE:

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

int curr = 0;
char str[100];

// Function declarations
void S();
void A();
void B();
```

```
void invalid();
void valid();
void match(char expected);
void skipSpaces();

// Error handling
void invalid() {
    printf("----------------ERROR!----------------\n");
    exit(0);
}

void valid() {
    printf("----------------SUCCESS!---------------\n");
    exit(0);
}

// Utility functions
void skipSpaces() {
    while (str[curr] == ' ' || str[curr] == '\t') {
        curr++;
    }
}

void match(char expected) {
    skipSpaces();
    if (str[curr] == expected) {
        curr++;
    } else {
        printf("Expected '%c' but found '%c'\n", expected, str[curr]);
        invalid();
    }
}
```

```
// Grammar rules

// S → aAcBe
void S() {
    skipSpaces();
    match('a');  // Match 'a'
    A();         // Parse A
    match('c');  // Match 'c'
    B();         // Parse B
    match('e');  // Match 'e'
}

// Original: A → Ab | b
// Transformed to: A → bA'
// A' → bA' | ε
void A() {
    skipSpaces();
    match('b');  // There must be at least one 'b'

    // Handle any additional b's (right recursion)
    while (str[curr] == 'b') {
        match('b');
    }
}

// B → d
void B() {
    skipSpaces();
    match('d');
}
```

```
int main() {
    printf("Enter String: ");
    fgets(str, 100, stdin);

    // Remove newline if present
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == '\n') {
            str[i] = '\0';
            break;
        }
    }

    S();  // Start parsing with S
    skipSpaces();

    if (str[curr] == '\0') {
        valid();
    } else {
        printf("Unexpected characters after valid expression: '%s'\n", &str[curr]);
        invalid();
    }

    return 0;
}
```

OUTPUT:

```
Enter String: abcde
----------SUCCESS!----------

Enter String: acde
Expected 'b' but found 'c'
----------ERROR!----------
```

Q4 - Write a recursive descent parser for the following simple grammars.

GRAMMAR -

S →(L) | a

L → L,S | S

Left Recursion: L → L, S | S

After left recursion removal:

S → (L) | a

L → S L'

L' → , S L' | ϵ

First Sets:

- First(S) = { (, a }
- First(L) = { (, a }
- First(L') = { , , ϵ }

Follow Sets:

- Follow(S) = {), $ }
- Follow(L) = {) }
- Follow(L') = {), $ }

CODE:

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

int curr = 0;
char str[100];

// Function declarations
void S();
void L();
void invalid();
void valid();
void match(char expected);
void skipSpaces();
```

```
// Error handling
void invalid() {
    printf("----------------ERROR!----------------\n");
    exit(0);
}

void valid() {
    printf("----------------SUCCESS!---------------\n");
    exit(0);
}

// Utility functions
void skipSpaces() {
    while (str[curr] == ' ' || str[curr] == '\t') {
        curr++;
    }
}

void match(char expected) {
    skipSpaces();
    if (str[curr] == expected) {
        curr++;
    } else {
        printf("Expected '%c' but found '%c'\n", expected, str[curr]);
        invalid();
    }
}

// Grammar rules

// S → (L) | a
void S() {
```

```
    skipSpaces();

    if (str[curr] == '(') {
        match('(');
        L();
        match(')');
    } else if (str[curr] == 'a') {
        match('a');
    } else {
        printf("Invalid symbol in S: '%c'. Expected '(' or 'a'\n", str[curr]);
        invalid();
    }
}

// Original: L → L,S | S
// Transformed to: L → S(,S)*
void L() {
    skipSpaces();

    // First S
    S();

    // Handle any additional ',S' pairs
    skipSpaces();
    while (str[curr] == ',') {
        match(',');
        S();
        skipSpaces();
    }
}

int main() {
```

```
    printf("Enter String: ");
    fgets(str, 100, stdin);

    // Remove newline if present
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == '\n') {
            str[i] = '\0';
            break;
        }
    }

    S();  // Start parsing with S
    skipSpaces();

    if (str[curr] == '\0') {
        valid();
    } else {
        printf("Unexpected characters after valid expression: '%s'\n", &str[curr]);
        invalid();
    }

    return 0;
}
```

OUTPUT:

```
Enter String: a
----------SUCCESS!----------
Enter String: ((a)
Expected ')' but found "
----------ERROR!----------
```