

COMPILER DESIGN LAB – 6

ADITYA AGARWAL

220905106

ROLL NO. 14

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

GRAMMAR -

$S \rightarrow a \mid > \mid (T)$

$T \rightarrow T, S \mid S$

Left Recursion: $T \rightarrow T, S \mid S$

After left recursion removal:

$S \rightarrow a \mid > \mid (T)$

$T \rightarrow S T'$

$T' \rightarrow , S T' \mid \epsilon$

First Sets:

- $\text{First}(S) = \{a, >, (\}$
- $\text{First}(T) = \{a, >, (\}$
- $\text{First}(T') = \{, , \epsilon\}$

Follow Sets:

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

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() {
    //  $S \rightarrow a \mid > \mid ( T )$ 
    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() {
    //  $T \rightarrow T, S \mid S$ 
    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 -

$S \rightarrow UVW$

$U \rightarrow (S) \mid aSb \mid d$

$V \rightarrow aV \mid \epsilon$

$W \rightarrow cW \mid \epsilon$

Left Recursion: NO

After left recursion removal:

SAME

First(S) = { (, a, d }

First(U) = { (, a, d }

First(V) = { a, ϵ }

First(W) = { c, ϵ }

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
```

```
//  $S \rightarrow UVW$ 
```

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

// $U \rightarrow (S) \mid aSb \mid d$

```
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();  
    }  
}
```

```
}
```

```
//  $V \rightarrow aV \mid$ 
```

```
void V() {
```

```
    skipSpaces();
```

```
    // Check if we have 'a', otherwise assume epsilon
```

```
    while (str[curr] == 'a') {
```

```
        match('a');
```

```
    }
```

```
    // epsilon production requires no action
```

```
}
```

```
//  $W \rightarrow cW \mid$ 
```

```
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 -

$S \rightarrow aAcBe$

$A \rightarrow Ab|b$

$B \rightarrow d$

Left Recursion: $A \rightarrow Ab \mid b$

After left recursion removal:

$S \rightarrow aAcBe$

$A \rightarrow bA'$

$A' \rightarrow bA' \mid \epsilon$

$B \rightarrow d$

First Sets:

- $\text{First}(S) = \{ a \}$
- $\text{First}(A) = \{ b \}$
- $\text{First}(A') = \{ b, \epsilon \}$
- $\text{First}(B) = \{ d \}$

Follow Sets:

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

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
```

```
//  $S \rightarrow aAcBe$ 
```

```
void S() {
```

```

    skipSpaces();
    match('a'); // Match 'a'
    A();       // Parse A
    match('c'); // Match 'c'
    B();       // Parse B
    match('e'); // Match 'e'
}

```

```

// Original:  $A \rightarrow Ab \mid b$ 
// Transformed to:  $A \rightarrow bA'$ 
//  $A' \rightarrow bA' \mid \epsilon$ 

```

```

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');
    }
}

```

```

//  $B \rightarrow d$ 
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 -

$S \rightarrow (L) \mid a$

$L \rightarrow L, S \mid S$

Left Recursion: $L \rightarrow L, S \mid S$

After left recursion removal:
$$S \rightarrow (L) \mid a$$
$$L \rightarrow S L'$$
$$L' \rightarrow , S L' \mid \epsilon$$
First Sets:

- $\text{First}(S) = \{ (, a \}$
- $\text{First}(L) = \{ (, a \}$
- $\text{First}(L') = \{ , , \epsilon \}$

Follow Sets:

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

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

//  $S \rightarrow (L) \mid a$ 
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: $L \rightarrow L,S \mid S$

// Transformed to: $L \rightarrow S(S)^*$

```

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!-----