CC LAB

Assignment 2

COMSATS UNIVERSITY

ATTOCK CAMPUS

Submitted By

SAFWANA

Registration No

SP21-BCS-028

Submitted To

SIR BILAL HAIDER

Date

17-03-2024

Tasks

Elevator control cfg and code with output:

Code

```
#include <iostream>
#include <stdlib.h>
using namespace std;
int count = 0;
string expr;
void E();
void Ed();
void T();
void Td();
void F();
int main() {
    cout << "Enter the elevator command sequence: ";
    cin >> expr;
    int l = expr.length();
    expr += "$"; // Append end of input marker
    if (expr[0] == 'D') {
        cout << "Rejected" << endl;
        return 0;
    }
    E(); // Start parsing
    if (l == count) {
        cout << "Accepted" << endl;
    } else {
        cout << "Rejected" << endl;
    }
```

```
    return 0;
}
void E() {
    cout << "E->TE'" << endl;
    T();
    Ed();
}
void Ed() {
    if (expr[count] == 'U') {
        count++;
        cout << "E'->UE'" << endl;
        T();
        Ed();
    } else if (expr[count] == 'D') {
        count++;
        cout << "E'->DE'" << endl;
        T();
        Ed();
    } else {
        cout << "E'->null" << endl;
    }
}
void T() {
    cout << "T->FT'" << endl;
    F();
    Td();
}
void Td() {
```

```
    if (expr[count] == 'U') {
        count++;
        cout << "T'->UT'" << endl;
        F();
        Td();
    } else if (expr[count] == 'D') {
        count++;
        cout << "T'->DT'" << endl;
        F();
        Td();
    } else {
        cout << "T'->null" << endl;
    }
}
void F() {
    if (expr[count] == 'U' || expr[count] == 'D') {
        count++;
        cout << "F->" << expr[count-1] << endl;
    } else {
        cout << "Rejected" << endl;
        exit(0);
    }
}
```

Output

1. **Up up up**

```
Output
/tmp/jjYngr4S96.o
Enter the elevator command sequence: UUU
E->TE'
T->FT'
F->U
T'->UT'
F->U
T'->null
E'->null
Accepted
```

2. **Up up down**

```
Output
/tmp/UCh8NslZl6.o
Enter the elevator command sequence: UUD
E->TE'
T->FT'
F->U
T'->UT'
F->D
T'->null
E'->null
Accepted
```

3. **Up down up**

```
Output
/tmp/XGpn3oLtmK.o
Enter the elevator command sequence: UDU
E->TE'
T->FT'
F->U
T'->DT'
F->U
T'->null
E'->null
Accepted
```

Lab 06 Task

Write a code for any given grammar that satisfy the criterion of JAVA language constructs.

Code

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>


#define MAX_TOKEN_LENGTH 100


typedef enum {
    TOK_INT,
    TOK_FLOAT,
    TOK_DOUBLE,
    TOK_IDENTIFIER,
    TOK_NUMBER,
    TOK_PLUS,
    TOK_MINUS,
    TOK_MULTIPLY,
    TOK_DIVIDE,
```

```
    TOK_ASSIGN,
    TOK_SEMICOLON,
    TOK_LPAREN,
    TOK_RPAREN,
    TOK_EOF,
    TOK_INVALID
} TokenType;

typedef struct {
    TokenType type;
    char value[MAX_TOKEN_LENGTH];
} Token;

Token current_token;

void next_token() {
    scanf("%s", current_token.value);

    if (strcmp(current_token.value, "int") == 0)
        current_token.type = TOK_INT;
    else if (strcmp(current_token.value, "float") == 0)
        current_token.type = TOK_FLOAT;
    else if (strcmp(current_token.value, "double") == 0)
        current_token.type = TOK_DOUBLE;
    else if (strcmp(current_token.value, "+") == 0)
        current_token.type = TOK_PLUS;
    else if (strcmp(current_token.value, "-") == 0)
        current_token.type = TOK_MINUS;
```

```
    else if (strcmp(current_token.value, "*") == 0)
        current_token.type = TOK_MULTIPLY;
    else if (strcmp(current_token.value, "/") == 0)
        current_token.type = TOK_DIVIDE;
    else if (strcmp(current_token.value, "=") == 0)
        current_token.type = TOK_ASSIGN;
    else if (strcmp(current_token.value, ";") == 0)
        current_token.type = TOK_SEMICOLON;
    else if (strcmp(current_token.value, "(") == 0)
        current_token.type = TOK_LPAREN;
    else if (strcmp(current_token.value, ")") == 0)
        current_token.type = TOK_RPAREN;
    else if (isalpha(current_token.value[0]))
        current_token.type = TOK_IDENTIFIER;
    else if (isdigit(current_token.value[0]))
        current_token.type = TOK_NUMBER;
    else
        current_token.type = TOK_INVALID;
}

void match(TokenType expected_token) {
    if (current_token.type == expected_token)
        next_token();
    else {
        printf("Syntax error: Expected %d, found %d\n", expected_token, current_token.type);
        exit(1);
    }
}
```

```
void expression();
void term();
void factor();

void program();

void declaration() {
    switch (current_token.type) {
        case TOK_INT:
        case TOK_FLOAT:
        case TOK_DOUBLE:
            next_token();
            match(TOK_IDENTIFIER);
            match(TOK_SEMICOLON);
            break;
        default:
            printf("Syntax error: Invalid declaration\n");
            exit(1);
    }
}

void assignment() {
    match(TOK_IDENTIFIER);
    match(TOK_ASSIGN);
    expression();
    match(TOK_SEMICOLON);
}
```

```
void factor() {
    switch (current_token.type) {
        case TOK_IDENTIFIER:
        case TOK_NUMBER:
            next_token();
            break;
        case TOK_LPAREN:
            match(TOK_LPAREN);
            expression();
            match(TOK_RPAREN);
            break;
        default:
            printf("Syntax error: Invalid factor\n");
            exit(1);
    }
}

void term() {
    factor();
    while (current_token.type == TOK_MULTIPLY || current_token.type == TOK_DIVIDE) {
        next_token();
        factor();
    }
}

void expression() {
    term();
```

```
    while (current_token.type == TOK_PLUS || current_token.type == TOK_MINUS) {
        next_token();
        term();
    }
}

void statement() {
    switch (current_token.type) {
        case TOK_INT:
        case TOK_FLOAT:
        case TOK_DOUBLE:
            declaration();
            break;
        case TOK_IDENTIFIER:
            assignment();
            break;
        default:
            printf("Syntax error: Invalid statement\n");
            exit(1);
    }
}

void statement_list() {
    while (current_token.type != TOK_EOF) {
        statement();
    }
}
```

```
void program() {
    statement_list();
}

int main() {
    next_token(); // Start parsing
    program(); // Start parsing the program
    printf("Parsing successful\n");
    return 0;
}
```