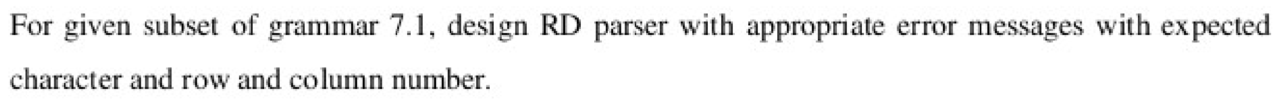
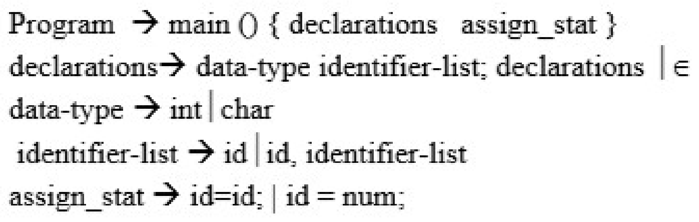
Saivya Singh
220905730
CSE D
44

Lab 7 : RD Parser for Declaration Statements

Q1 .


For given subset of grammar 7.1, design RD parser with appropriate error messages with expected character and row and column number.


Program → main () { declarations assign_stat }
declarations→ data-type identifier-list; declarations | ∈
data-type → int | char
identifier-list → id | id, identifier-list
assign_stat → id=id; | id = num;

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
typedef enum { T_MAIN, T_INT, T_CHAR, T_ID, T_NUM, T_ASSIGN,
T_SEMICOLON, T_COMMA, T_LPAREN, T_RPAREN, T_LBRACE, T_RBRACE, T_EOF,
T_UNKNOWN } TokenType;
typedef struct { TokenType type; char lexeme[128]; int line; int col; } Token;
static Token currentToken;
static int g_line = 1;
static int g_col = 0;
static FILE* source;
void parseProgram();
void parseDeclarations();
void parseDataType();
void parseIdentifierList();
void parseAssignStat();
const char* tokenTypeName(TokenType t) {
    switch(t) {
        case T_MAIN: return "main";
        case T_INT: return "int";
        case T_CHAR: return "char";
        case T_ID: return "identifier";
        case T_NUM: return "number";
        case T_ASSIGN: return "=";
        case T_SEMICOLON: return ";";
        case T_COMMA: return ",";
        case T_LPAREN: return "(";
        case T_RPAREN: return ")";
        case T_LBRACE: return "{";
        case T_RBRACE: return "}";
        case T_EOF: return "EOF";
        default: return "UNKNOWN";
    }
}
void error(const char* expected) {
    fprintf(stderr, "Error at line %d, col %d: expected %s but found '%s' (token
type: %s)\n", currentToken.line, currentToken.col, expected,
currentToken.lexeme, tokenTypeName(currentToken.type));
    exit(EXIT_FAILURE);
}
int nextChar() {
    int c = fgetc(source);
    if(c == '\n') { g_line++; g_col = 0; } else { g_col++; }
    return c;
}
int peekChar() {
    int c = fgetc(source);
    ungetc(c, source);
    return c;
}
int skipWhitespace() {
    int c;
    do { c = nextChar(); } while(isspace(c));
    return c;
}
Token getNextToken() {
    Token token;
    int c;
    token.type = T_UNKNOWN;
    token.lexeme[0] = '\0';
    token.line = g_line;
    token.col = g_col;
    c = skipWhitespace();
    if(c == EOF) { token.type = T_EOF; strcpy(token.lexeme, "EOF"); return
token; }
    token.line = g_line;
    token.col = g_col;
    if(c == '(') { token.type = T_LPAREN; strcpy(token.lexeme, "("); return token;
}
    if(c == ')') { token.type = T_RPAREN; strcpy(token.lexeme, ")"); return token;
}
    if(c == '{') { token.type = T_LBRACE; strcpy(token.lexeme, "{"); return
token; }
    if(c == '}') { token.type = T_RBRACE; strcpy(token.lexeme, "}"); return
token; }
    if(c == '=') { token.type = T_ASSIGN; strcpy(token.lexeme, "="); return
token; }
    if(c == ';') { token.type = T_SEMICOLON; strcpy(token.lexeme, ";"); return
token; }
    if(c == ',') { token.type = T_COMMA; strcpy(token.lexeme, ","); return
token; }
```

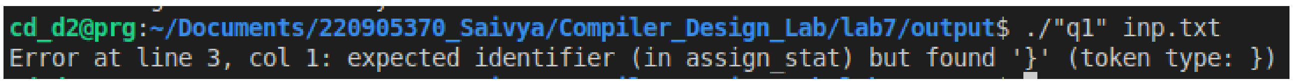
```
    if(isalpha(c)) {
        char buffer[128];
        int i = 0;
        buffer[i++] = (char)c;
        while(isalnum(peekChar()) || peekChar() == '_') { c = nextChar();
buffer[i++] = (char)c; }
        buffer[i] = '\0';
        if(strcmp(buffer, "main") == 0) token.type = T_MAIN;
        else if(strcmp(buffer, "int") == 0) token.type = T_INT;
        else if(strcmp(buffer, "char") == 0) token.type = T_CHAR;
        else token.type = T_ID;
        strcpy(token.lexeme, buffer);
        return token;
    }
    if(isdigit(c)) {
        char buffer[128];
        int i = 0;
        buffer[i++] = (char)c;
        while(isdigit(peekChar())) { c = nextChar(); buffer[i++] = (char)c; }
        buffer[i] = '\0';
        token.type = T_NUM;
        strcpy(token.lexeme, buffer);
        return token;
    }
    token.type = T_UNKNOWN;
    token.lexeme[0] = (char)c;
    token.lexeme[1] = '\0';
    return token;
}
void advance() { currentToken = getNextToken(); }
void match(TokenType expected) { if(currentToken.type == expected)
{ advance(); } else { error(tokenTypeName(expected)); } }
void parseProgram() {
    match(T_MAIN);
    match(T_LPAREN);
    match(T_RPAREN);
    match(T_LBRACE);
    parseDeclarations();
    parseAssignStat();
    match(T_RBRACE);
    printf("Parsed Program successfully.\n");
}
void parseDeclarations() {
    if(currentToken.type == T_INT || currentToken.type == T_CHAR) {
        parseDataType();
        parseIdentifierList();
        match(T_SEMICOLON);
        parseDeclarations();
    }
}
void parseDataType() {
    if(currentToken.type == T_INT) { match(T_INT); }
    else if(currentToken.type == T_CHAR) { match(T_CHAR); }
    else { error("int or char"); }
}
void parseIdentifierList() {
    if(currentToken.type == T_ID) { match(T_ID); }
    else { error("identifier"); }
    if(currentToken.type == T_COMMA) { match(T_COMMA); parseIdentifierList();
}
}
void parseAssignStat() {
    if(currentToken.type == T_ID) {
        match(T_ID);
        match(T_ASSIGN);
        if(currentToken.type == T_ID) { match(T_ID); }
        else if(currentToken.type == T_NUM) { match(T_NUM); }
        else { error("identifier or number"); }
        match(T_SEMICOLON);
    } else { error("identifier (in assign_stat)"); }
}
int main(int argc, char* argv[]) {
    if(argc < 2) { printf("Usage: %s <source-file>\n", argv[0]); return 1; }
    source = fopen(argv[1], "r");
    if(!source) { perror("Error opening file"); return 1; }
    advance();
    parseProgram();
    if(currentToken.type != T_EOF) { error("EOF"); }
    fclose(source);
    return 0;
}
```

Input :

```
main() {

}
```

Output :

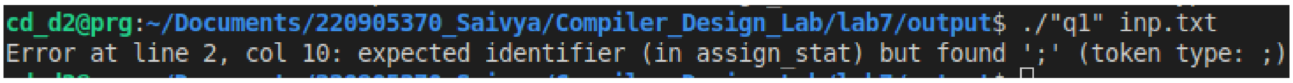


```
cd_d2@prg:~/Documents/220905370_Saivya/Compiler_Design_Lab/lab7/output$ ./"q1" inp.txt
Error at line 3, col 1: expected identifier (in assign_stat) but found '}' (token type: })
```

Input :

```
main(){
    int a;
}
```

Output :

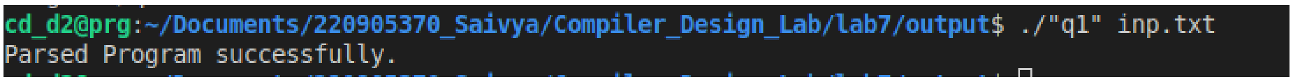


```
cd_d2@prg:~/Documents/220905370_Saivya/Compiler_Design_Lab/lab7/output$ ./"q1" inp.txt
Error at line 2, col 10: expected identifier (in assign_stat) but found ';' (token type: ;)
```

Input :

```
main(){
   int a,b;
   a=5;
}
```

Output :

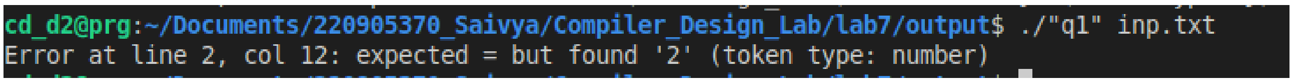


```
cd_d2@prg:~/Documents/220905370_Saivya/Compiler_Design_Lab/lab7/output$ ./"q1" inp.txt
Parsed Program successfully.
```

Input :

```
main(){
   return 2;
}
```

Output :

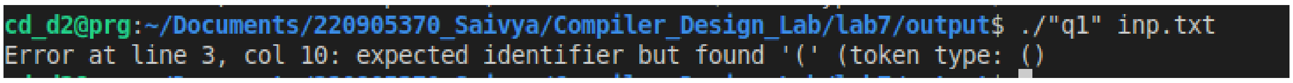


```
cd_d2@prg:~/Documents/220905370_Saivya/Compiler_Design_Lab/lab7/output$ ./"q1" inp.txt
Error at line 2, col 12: expected = but found '2' (token type: number)
```

Input :

```
main(){
   int a,b;
   char (i);
   i = b+c;
   a=10;
   c=a;
}
```

Output :



```
cd_d2@prg:~/Documents/220905370_Saivya/Compiler_Design_Lab/lab7/output$ ./"q1" inp.txt
Error at line 3, col 10: expected identifier but found '(' (token type: ()
```