CD Lab-7 Assignment

Name: Ketan Goud
Reg No: 220905260
Section: D D2
Roll No: 39

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() { declaration assign_stat }
declarations -> data-type identifier-list; declarations | E
data-type -> int | char
identifier_list -> id | id, identifier-list
assign_stat -> id=id | id=num;

Program -> main() { declarations assign_stat }
declarations -> data_type identifier_list; declarations | E
data_type -> int | char
identifier_list -> idA'
A' -> ,identifier_list | E
assign_stat -> id=B';
B' -> id | num

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

void program();
void declarations();
void data_type(char *type);
void identifier_list();
void A_prime();
void assign_stat();
void B_prime();

struct token
{
   char token[50];
   int row;
   int col;
   int index;
   char type[20];
};
struct token table[200];
struct symbol
{
   int sno;
   char lexeme[50];
   char datatype[20];
   int size;
};
struct symbol symtable[20];
int symind=0;
struct function {
   int sno;
   char function[20];
   char returntype[20];
   char parameters[50];
   int num;
};
struct function functable[20];
int funcind=0;

const char *keywords[] = {
   "int", "float", "double", "char", "if", "else", "for", "return",
   "while", "void", "switch", "case", "break", "continue",
   "default", "struct", "union", "enum", "long", "short", "const",
   "sizeof", "printf", "scanf"
};

#define NUM_KEYWORDS 24

int iskeyword(char *word)
{
   for(int i=0;i<NUM_KEYWORDS;i++)
   {
       if(strcmp(word,keywords[i])==0)
       {
           return 1;
       }
   }
   return 0;
}

int tokenexists(char *word, int ind)
{
   for(int i=0;i<ind;i++)
   {
       if(strcmp(word, table[i].token)==0)
       {
           return table[i].index;
       }
   }
   return -1;
}

int getdatatypesize(char *word)
{
   if(strcmp(word,"int")==0) return 4;
   else if(strcmp(word,"float")==0) return 4;
   else if(strcmp(word,"double")==0) return 8;
   else if(strcmp(word,"char")==0) return 1;
   else return 0;
}

void addtotable(char *lexeme, char *datatype, int arraysize)
{
   int size = getdatatypesize(datatype) * arraysize;
   for(int i=0; i<symind; i++)
   {
       if(strcmp(lexeme, symtable[i].lexeme) == 0)
       {
           return;
       }
   }
   symtable[symind].sno = symind + 1;
   strcpy(symtable[symind].lexeme, lexeme);
   strcpy(symtable[symind].datatype, datatype);
   symtable[symind].size = size;
   symind++;
}

void addtofunctable(char *function, char *returntype, char *parameters, int num)
{
   for(int i=0; i<funcind; i++)
   {
       if(strcmp(function, functable[i].function) == 0)
       {
           return;
       }
   }
   functable[funcind].sno= funcind+1;
   strcpy(functable[funcind].function,function);
   strcpy(functable[funcind].returntype,returntype);
   strcpy(functable[funcind].parameters,parameters);
   functable[funcind].num=num;
   funcind++;
}

void get_token(FILE *f1, FILE *f2)
{
   char word[20];
   char datatype[20];
   char number[20];
   char string[30];
   char returntype[20];
   char parameters[30] = "";
   char c, next;
   int i = 0;
   int row = 1, col = 1;
   int cur_row, cur_col;
   int isfunc = 0;
   int arraysize = 1;
   int ind = 0;

   while ((c = getc(f1)) != EOF)
   {
       if (isspace(c))
       {
           if (c == '\n')
           {
               row++;
               col = 1;
           }
           else
           {
               col++;
           }
           continue;
       }
       cur_col = col;
       cur_row = row;

       if (c == '+' || c == '-' || c == '*' || c == '/' || c == '%')
       {
           table[ind].index = ind + 1;
           table[ind].row = cur_row;
           table[ind].col = cur_col;
           table[ind].token[0] = c;
           table[ind].token[1] = '\0';
           strcpy(table[ind].type, "Arithmetic Op");
           fprintf(f2, "<%d, '%s', %d, %d, '%s'>\n", table[ind].index, table[ind].token, table[ind].row,
table[ind].col, table[ind].type);
           ind++;
           col++;
           continue;
       }

       if (c == '=' || c == '>' || c == '<' || c == '!')
       {
           next = getc(f1);
           if (next == '=')
           {
               col++;
               table[ind].index = ind + 1;
               table[ind].row = cur_row;
               table[ind].col = cur_col;
               table[ind].token[0] = c;
               table[ind].token[1] = next;
               table[ind].token[2] = '\0';
               strcpy(table[ind].type, "Relational Op");
               fprintf(f2, "<%d, '%s', %d, %d, '%s'>\n", table[ind].index, table[ind].token, table[ind].row,
table[ind].col, table[ind].type);
               ind++;
               col++;
           }
           else
           {
               ungetc(next, f1);
               if (c == '=')
               {
                   table[ind].index = ind + 1;
                   table[ind].row = cur_row;
                   table[ind].col = cur_col;
                   table[ind].token[0] = c;
                   table[ind].token[1] = '\0';
                   strcpy(table[ind].type, "Assignment Op");
                   fprintf(f2, "<%d, '%s', %d, %d, '%s'>\n", table[ind].index, table[ind].token, table[ind].row,
table[ind].col, table[ind].type);
                   ind++;
                   col++;
               }
               else
               {
                   table[ind].index = ind + 1;
                   table[ind].row = cur_row;
                   table[ind].col = cur_col;
                   table[ind].token[0] = c;
                   table[ind].token[1] = '\0';
                   strcpy(table[ind].type, "Relational Op");
                   fprintf(f2, "<%d, '%s', %d, %d, '%s'>\n", table[ind].index, table[ind].token, table[ind].row,
table[ind].col, table[ind].type);
                   ind++;
                   col++;
               }
           }
           continue;
       }

       if (isdigit(c))
       {
           i = 0;
           number[i] = c;
           i++;
           col++;
           c = getc(f1);
           while (isdigit(c) || c == '.')
           {
               number[i] = c;
               i++;
               c = getc(f1);
           }
           number[i] = '\0';
           ungetc(c, f1);
           strcpy(table[ind].token, number);
           strcpy(table[ind].type, "Numeric");
           table[ind].index = ind + 1;
           table[ind].row = cur_row;
           table[ind].col = cur_col;
           fprintf(f2, "<%d, '%s', %d, %d, '%s'>\n", table[ind].index, table[ind].token, table[ind].row,
table[ind].col, table[ind].type);
           ind++;
           col += i;
           continue;
       }

       if (isalpha(c) || c == '_')
       {
           i = 0;
           word[i] = c;
           i++;
           c = getc(f1);
           col++;
           while (isalnum(c) || c == '_')
           {
               word[i] = c;
               i++;
               c = getc(f1);
           }
           word[i] = '\0';
           ungetc(c, f1);
           strcpy(table[ind].token, word);
```

```
            table[ind].index = ind + 1;
            table[ind].row = cur_row;
            table[ind].col = cur_col;
            if (iskeyword(word))
            {
                strcpy(table[ind].type, "Keyword");
                fprintf(f2, "<%d, '%s', %d, %d, '%s'>\n", table[ind].index, table[ind].token, table[ind].row,
table[ind].col, table[ind].type);
                ind++;
                strcpy(datatype, word);
            }
            else
            {
                strcpy(table[ind].type, "Identifier");
                arraysize = 1;
                c = getc(f1);
                if (c == '[')
                {
                    fscanf(f1, "%d]", &arraysize);
                }
                else
                {
                    ungetc(c, f1);
                }
                if (datatype[0] != '\0')
                {
                    addtotable(word, datatype, arraysize);
                }
                fprintf(f2, "<%d, '%s', %d, %d, '%s'>\n", table[ind].index, table[ind].token, table[ind].row,
table[ind].col, table[ind].type);
                ind++;
            }
            col += i;
            continue;
        }

        if (c == '"')
        {
            i = 0;
            string[i] = c;
            i++;
            col++;
            c = getc(f1);
            while (c != '"')
            {
                string[i] = c;
                i++;
                c = getc(f1);
            }
            string[i] = '"';
            i++;
            string[i] = '\0';
            strcpy(table[ind].token, string);
            strcpy(table[ind].type, "String Literal");
            table[ind].index = ind + 1;
            table[ind].row = cur_row;
            table[ind].col = cur_col;
            fprintf(f2, "<%d, '%s', %d, %d, '%s'>\n", table[ind].index, table[ind].token, table[ind].row,
table[ind].col, table[ind].type);
            ind++;
            col += i;
            continue;
        }

        if (c == '(' || c == ')' || c == '[' || c == ']' || c == '{' || c == '}')
        {
            table[ind].index = ind + 1;
            table[ind].row = cur_row;
            table[ind].col = cur_col;
            table[ind].token[0] = c;
            table[ind].token[1] = '\0';
            strcpy(table[ind].type, "Parenthesis");
            fprintf(f2, "<%d, '%s', %d, %d, '%s'>\n", table[ind].index, table[ind].token, table[ind].row,
table[ind].col, table[ind].type);
            ind++;
            col++;
            continue;
        }
        if (c == ',')
        {
            table[ind].index = ind + 1;
            table[ind].row = cur_row;
            table[ind].col = cur_col;
            table[ind].token[0] = c;
            table[ind].token[1] = '\0';
            strcpy(table[ind].type, "colon");
            fprintf(f2, "<%d, '%s', %d, %d, '%s'>\n", table[ind].index, table[ind].token, table[ind].row,
table[ind].col, table[ind].type);
            ind++;
            col++;

        }

        if (c == ';')
        {
            table[ind].index = ind + 1;
            table[ind].row = cur_row;
            table[ind].col = cur_col;
            table[ind].token[0] = c;
            table[ind].token[1] = '\0';
            strcpy(table[ind].type, "semi colon");
            fprintf(f2, "<%d, '%s', %d, %d, '%s'>\n", table[ind].index, table[ind].token, table[ind].row,
table[ind].col, table[ind].type);
            ind++;
            col++;
            continue;
        }
    }
}

int current_index = 1;

struct token get_next_token()
{
    return table[current_index++];
}
void data_type(char *type)
{
    if (strcmp(type, "int") == 0)
    {
        printf("Parsed data type: int\n");
    }
    else if (strcmp(type, "char") == 0)
    {
        printf("Parsed data type: char\n");
    }
    else {
        printf("Error: Unexpected data type\n");
    }
}
void program()
{
    struct token t = get_next_token();
    if (strcmp(t.token, "main") == 0)
    {
        t = get_next_token();
        if (strcmp(t.token, "(") == 0)
        {
            t = get_next_token();
            if (strcmp(t.token, ")") == 0)
            {
                t = get_next_token();
                if (strcmp(t.token, "{") == 0)
                {
                    declarations();
                    assign_stat();
                    t = get_next_token();
                    if (strcmp(t.token, "}") != 0)
                    {
                        printf("Error: Expected '}'\n");
                    }
                }
                else
                {
                    printf("Error: Expected '{'\n");
                }
            }
            else
            {
                printf("Error: Expected ')'\n");
            }
        }
        else
        {
            printf("Error: Expected '('\n");
        }
    }
    else
    {
        printf("Error: Expected 'main'\n");
    }
```

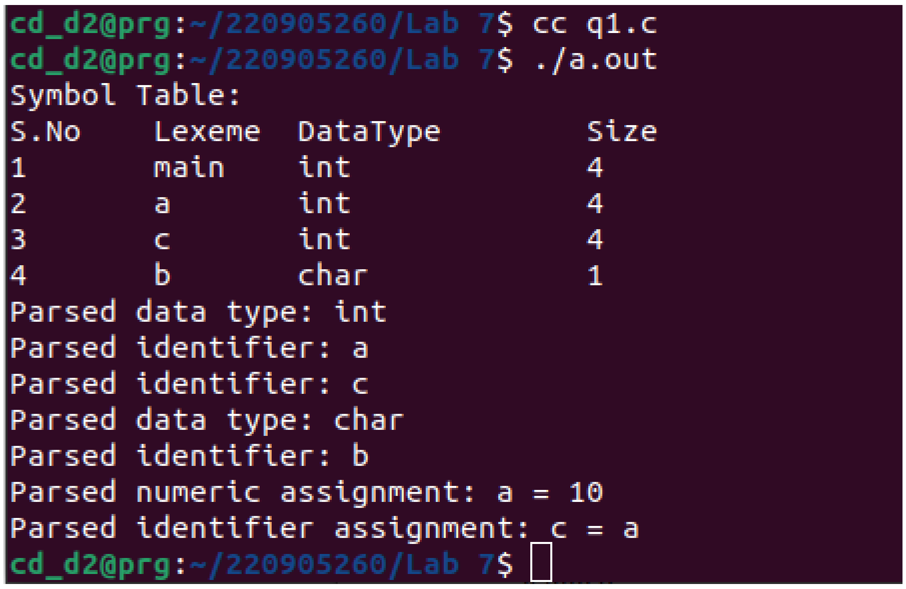
```
}

void declarations()
{
   struct token t = get_next_token();
   if (strcmp(t.token, "int") == 0 || strcmp(t.token, "char") == 0)
   {
      data_type(t.token);
      identifier_list();
      t = get_next_token();
      if (strcmp(t.token, ";") == 0)
      {
         declarations();
      }
      else
      {
         printf("Error: Expected ';' after declaration\n");
      }
   }
   else
   {
      current_index--;
   }
}

void identifier_list()
{
   struct token t = get_next_token();
   if (strcmp(t.type, "Identifier") == 0)
   {
      A_prime(t);
      t = get_next_token();
      if (strcmp(t.token, ",") == 0)
      {
         identifier_list();
      }
      else
      {
         current_index--;
      }
   }
   else
   {
      printf("Error: Expected identifier in identifier list\n");
   }
}

void A_prime(struct token t)
{
   if (strcmp(t.type, "Identifier") == 0)
   {
      printf("Parsed identifier: %s\n", t.token);
   }
   else
   {
      printf("Error: Expected identifier\n");
   }
```

```
}

void assign_stat()
{
    while (1)
    {
        struct token t = get_next_token();

        if (strcmp(t.type, "Identifier") == 0)
        {
            B_prime(t);

            struct token t_end = get_next_token();
            if (strcmp(t_end.token, ";") == 0)
            {
                continue;
            }
            else
            {
                printf("Error: Expected ';' after assignment\n");
                return;
            }
        }
        else
        {
            current_index--;
            break;
        }
    }
}

void B_prime(struct token t)
{
    struct token t_next = get_next_token();

    if (strcmp(t_next.token, "=") == 0)
    {
        struct token t_value = get_next_token();

        if (strcmp(t_value.type, "Numeric") == 0)
        {
            printf("Parsed numeric assignment: %s = %s\n", t.token, t_value.token);
        }
        else if (strcmp(t_value.type, "Identifier") == 0)
        {
            printf("Parsed identifier assignment: %s = %s\n", t.token, t_value.token);
        }
        else
        {
            printf("Error: Expected a numeric value or identifier after '='\n");
        }
    }
    else
    {
        printf("Error: Expected '=' in assignment\n");
    }
}
```

```
int main()
{
    FILE *f1 = fopen("s1.txt", "r");
    FILE *f2 = fopen("s2.txt", "w");
    get_token(f1, f2);

    printf("Symbol Table:\n");
    printf("S.No\tLexeme\tDataType\tSize\n");
    for (int i = 0; i < symind; i++)
    {
        printf("%d\t%s\t%s\t\t%d\n", symtable[i].sno, symtable[i].lexeme, symtable[i].datatype,
symtable[i].size);
    }
    program();
    fclose(f1);
    fclose(f2);
    return 0;
}
```

Sample C Code:

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



```
cd_d2@prg:~/220905260/Lab 7$ cc q1.c
cd_d2@prg:~/220905260/Lab 7$ ./a.out
Symbol Table:
S.No    Lexeme  DataType        Size
1       main    int             4
2       a       int             4
3       c       int             4
4       b       char            1
Parsed data type: int
Parsed identifier: a
Parsed identifier: c
Parsed data type: char
Parsed identifier: b
Parsed numeric assignment: a = 10
Parsed identifier assignment: c = a
cd_d2@prg:~/220905260/Lab 7$ 
```