CD Lab #8

Question 1

parser.c

```
/*

Design the recursive descent parser to parse array declarations and expression
statements with error reporting. Subset of grammar 7.1 is as follows:

Program -> main ( ) { Declarations Statement-List }
Declarations -> Data-Type Identifier-List ; Declarations | EP

Data-Type -> int | char

Identifier-List -> id Identifier-List-prime
Identifier-List-prime -> , Identifier-List | [ number ] Identifier-List-prime-prime | EP
Identifier-List-prime-prime -> , Identifier-List | EP

Statement-List -> Statement Statement-List | EP
Statement -> Assign-Stat | Decision_Stat | Looping_Stat;
Assign-Stat -> id = Expn

Expn -> Simple-Expn Eprime
Eprime -> Relop Simple-Expn | EP
Simple-Expn -> Term SEprime

SEprime -> Addop Term SEprime | EP
Term -> Factor Tprime
Tprime -> Mulop Factor Tprime | EP
Factor -> id | num

Decision_Stat -> if ( Expn ) { Statement_List } Dprime
Dprime -> else { Statement_List } | EP
Looping_Stat -> while ( Expn ) { Statement_List } | for ( Assign-Stat ; Expn ; Assign-Stat )
{ Statement_List }

Relop -> == | != | <= | >= | > | <
Addop -> + | -
Mulop -> * | / | %

*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "la.h"

FILE *f;
struct token t;
```

```
void printToken(){
    FILE *fd;
    struct token tkn;
    int numTkn=200;

    fd = fopen("input.c", "r");

    if (fd == NULL)
    {
        printf("Cannot open file\n");
        exit(0);
    }

    int i;
    for(i=0; i<=numTkn; i++){
        tkn = getNextToken(fd);
        if(tkn.row == -1)
            exit(0);
        printf("%d. <%s, %d, %d, %s>\n", i, tkn.lexeme, tkn.row, tkn.col, tkn.type);
    }
}

void valid()
{
    printf("Success\n\n\n=== Tokens Generated are ===\n");
    printToken();
    exit(0);
}

void invalid()
{
    printf("at position (row : %d, col : %d)\n\n", t.row, (t.col-6));
    printf("ERROR\n");
    exit(0);
}

void Program();       //done
void declarations();  //done
void datatype();      //done
void id_list();       //done
void id_listPrime();  //done
void id_listPrime2(); //done
void stmt_list();     //done
void statement();     //done
void assign_stat();   //done
void expn();          //done
void eprime();        //done
void simple_expn();   //done
void seprime();       //done
void term();          //done
void tprime();        //done
void factor();        //done
```

```
void decision_stat(); //done
void dprime();        //done
void looping_stat();  //done
void relop();         //done
void addop();         //done
void mulop();         //done

void Program()
{
    t = getNextToken(f);

    if (strcmp(t.lexeme, "main") == 0)
    {
        t = getNextToken(f);

        if (strcmp(t.lexeme, "(") == 0)
        {
            t = getNextToken(f);
            if (strcmp(t.lexeme, ")") == 0)
            {
                t = getNextToken(f);
                if (strcmp(t.lexeme, "{") == 0)
                {
                    declarations();
                    stmt_list();
                    t = getNextToken(f);
                    if (strcmp(t.lexeme, "}") == 0)
                    {
                        valid();
                    }
                    else
                    {
                        printf("Expected }\n");
                        invalid();
                    }
                }
                else
                {
                    printf("Expected {\n");
                    invalid();
                }
            }
            else
            {
                printf("Expected LC\n");
                invalid();
            }
        }
        else
        {
            printf("Expected RC\n");
            invalid();
```

```
        }
    }
    else
    {
        printf("Expected main\n");
        invalid();
    }
}

void declarations()
{
    t = getNextToken(f);
    int len = strlen(t.lexeme);
    if (strcmp(t.type, "identifier") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.type, "RC") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else
        fseek(f, -len, SEEK_CUR);

    datatype();
    id_list();
    t = getNextToken(f);
    numTkn++;
    if (strcmp(t.lexeme, ";") == 0)
    {
        declarations();
        return;
    }
    else if (strcmp(t.type, "identifier") == 0)
    {
        fseek(f, -1, SEEK_CUR);
        return;
    }
    else
    {
        printf("Expected ; or identifier\n");
        invalid();
    }
}

void datatype()
{
    t = getNextToken(f);
    if (strcmp(t.type, "keyword") == 0)
        return;
```

```
    else
    {
        printf("Expected int or char\n");
        invalid();
    }
}

void id_list()
{
    t = getNextToken(f);
    if (strcmp(t.type, "identifier") == 0)
    {
        id_listPrime();
        return;
    }
    else
    {
        printf("Expected identifier\n");
        invalid();
    }
}

void id_listPrime()
{
    t = getNextToken(f);
    int len = strlen(t.lexeme);
    if (strcmp(t.lexeme, ";") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else
        fseek(f, -len, SEEK_CUR);

    t = getNextToken(f);
    if (strcmp(t.lexeme, ",") == 0)
    {
        id_list();
        return;
    }
    else if (strcmp(t.lexeme, "[") == 0)
    {
        t = getNextToken(f);

        if (strcmp(t.type, "number") == 0)
        {
            t = getNextToken(f);

            if (strcmp(t.lexeme, "]") == 0)
            {
                id_listPrime2();
                return;
```

```
            }
            else
            {
                printf("Expected ]\n");
                invalid();
            }
        }
        else
        {
            printf("Expected number\n");
            invalid();
        }
    }
    else
    {
        printf("Expected , or [\n");
        invalid();
    }
}

void id_listPrime2()
{
    t = getNextToken(f);
    int len = strlen(t.lexeme);
    if (strcmp(t.lexeme, ";") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else
        fseek(f, -len, SEEK_CUR);

    t = getNextToken(f);
    if (strcmp(t.lexeme, ",") == 0)
    {
        id_list();
        return;
    }
    else
    {
        printf("Expected ,\n");
        invalid();
    }
}

void stmt_list()
{
    t = getNextToken(f);
    int len = strlen(t.lexeme);
    if (strcmp(t.type, "RC") == 0)
    {
        fseek(f, -len, SEEK_CUR);
```

```
        return;
    }
    else
        fseek(f, -len, SEEK_CUR);

    statement();
    stmt_list();
    return;
}

void statement()
{
    t = getNextToken(f);
    int len = strlen(t.lexeme);
    if (strcmp(t.lexeme, "if") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        decision_stat();
        return;
    }
    else if (strcmp(t.lexeme, "while") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        looping_stat();
        return;
    }
    else if (strcmp(t.lexeme, "for") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        looping_stat();
        return;
    }
    else
    {
        fseek(f, -len, SEEK_CUR);
        assign_stat();
        t = getNextToken(f);

        if (strcmp(t.lexeme, ";") == 0)
            return;
        else
        {
            printf("Expected ;\n");
            invalid();
        }
    }
}

void assign_stat()
{
    t = getNextToken(f);
    if (strcmp(t.type, "identifier") == 0)
```

```
    {
        t = getNextToken(f);

        if (strcmp(t.lexeme, "=") == 0)
        {
            expn();
            return;
        }
        else
        {
            printf("Expected =\n");
            invalid();
        }
    }
    else
    {
        printf("Expected identifier\n");
        invalid();
    }
}

void expn()
{
    simple_expn();
    eprime();
    return;
}

void eprime()
{
    t = getNextToken(f);
    int len = strlen(t.lexeme);
    if (strcmp(t.lexeme, ";") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, ")") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else
        fseek(f, -len, SEEK_CUR);

    relop();
    simple_expn();
    return;
}

void simple_expn()
{
```

```
    term();
    seprime();
    return;
}

void seprime()
{
    t = getNextToken(f);
    int len = strlen(t.lexeme);
    if (strcmp(t.lexeme, ";") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, "==") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, "!=") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, "<=") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, ">=") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, "<") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, ">") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, ")") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else
        fseek(f, -len, SEEK_CUR);
```

```
    addop();
    term();
    seprime();
    return;
}

void term()
{
    factor();
    tprime();
    return;
}

void tprime()
{
    t = getNextToken(f);
    int len = strlen(t.lexeme);
    if (strcmp(t.lexeme, ";") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, "==") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, "!=") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, "<=") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, ">=") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, ">") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, "<") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
```

```
    else if (strcmp(t.lexeme, "+") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, "-") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else if (strcmp(t.lexeme, ")") == 0)
    {
        fseek(f, -len, SEEK_CUR);
        return;
    }
    else
        fseek(f, -len, SEEK_CUR);

    mulop();
    factor();
    tprime();
    return;
}

void factor()
{
    t = getNextToken(f);
    if (strcmp(t.type, "identifier") == 0)
        return;
    else if (strcmp(t.type, "number") == 0)
        return;
    else
    {
        printf("Expected identifier or number\n");
        invalid();
    }
}

void decision_stat()
{
    t = getNextToken(f);
    if (strcmp(t.lexeme, "if") == 0)
    {
        t = getNextToken(f);

        if (strcmp(t.lexeme, "(") == 0)
        {
            expn();
            t = getNextToken(f);

            if (strcmp(t.lexeme, ")") == 0)
            {
```

```
            t = getNextToken(f);

            if (strcmp(t.lexeme, "{") == 0)
            {
                stmt_list();
                t = getNextToken(f);

                if (strcmp(t.lexeme, "}") == 0)
                {
                    dprime();
                    return;
                }
                else
                {
                    printf("Expected }\n");
                    invalid();
                }
            }
            else
            {
                printf("Expected {\n");
                invalid();
            }
        }
        else
        {
            printf("Expected )\n");
            invalid();
        }
    }
    else
    {
        printf("Expected (\n");
        invalid();
    }
}
else
{
    printf("Expected if\n");
    invalid();
}
}

void dprime()
{
    //follow remaining
    t = getNextToken(f);
    if (strcmp(t.lexeme, "else") == 0)
    {
        t = getNextToken(f);

        if (strcmp(t.lexeme, "{") == 0)
```

```
        {
            stmt_list();
            t = getNextToken(f);

            if (strcmp(t.lexeme, "}") == 0)
                return;
            else
            {
                printf("Expected }\n");
                invalid();
            }
        }
        else
        {
            printf("Expected {\n");
            invalid();
        }
    }
    else
    {
        printf("Expected else\n");
        invalid();
    }
}

void looping_stat()
{
    t = getNextToken(f);
    if (strcmp(t.lexeme, "while") == 0)
    {
        t = getNextToken(f);

        if (strcmp(t.lexeme, "(") == 0)
        {
            expn();
            t = getNextToken(f);

            if (strcmp(t.lexeme, ")") == 0)
            {
                t = getNextToken(f);

                if (strcmp(t.lexeme, "{") == 0)
                {
                    stmt_list();
                    t = getNextToken(f);

                    if (strcmp(t.lexeme, "}") == 0)
                        return;
                    else
                    {
                        printf("Expected }\n");
                        invalid();
```

```
            }
        }
        else
        {
            printf("Expected {\n");
            invalid();
        }
    }
    else
    {
        printf("Expected )\n");
        invalid();
    }
}
else
{
    printf("Expected (\n");
    invalid();
}
}
else if (strcmp(t.lexeme, "for") == 0)
{
    t = getNextToken(f);\

    if (strcmp(t.lexeme, "(") == 0)
    {
        assign_stat();

        t = getNextToken(f);
        if (strcmp(t.lexeme, ";") == 0)
        {
            expn();
            t = getNextToken(f);

            if (strcmp(t.lexeme, ";") == 0)
            {
                assign_stat();
                t = getNextToken(f);

                if (strcmp(t.lexeme, ")") == 0)
                {
                    t = getNextToken(f);

                    if (strcmp(t.lexeme, "{") == 0)
                    {
                        stmt_list();
                        t = getNextToken(f);

                        if (strcmp(t.lexeme, "}") == 0)
                            return;
                        else
                        {
```

```
                    printf("Expected }\n");
                    invalid();
                }
            }
            else
            {
                printf("Expected {\n");
                invalid();
            }
        }
        else
        {
            printf("Expected )\n");
            invalid();
        }
    }
    else
    {
        printf("Expected ;\n");
        invalid();
    }
}
else
{
    printf("Expected ;\n");
    invalid();
}
}
else
{
    printf("Expected (\n");
    invalid();
}
}
else
{
    printf("Expected while or for\n");
    invalid();
}
}

void relop()
{
    t = getNextToken(f);
    if (strcmp(t.lexeme, "==") == 0)
        return;
    else if (strcmp(t.lexeme, "!=") == 0)
        return;
    else if (strcmp(t.lexeme, "<") == 0)
        return;
    else if (strcmp(t.lexeme, ">") == 0)
        return;
```

```
    else if (strcmp(t.lexeme, "<=") == 0)
        return;
    else if (strcmp(t.lexeme, ">=") == 0)
        return;
    else
    {
        printf("Expected relop\n");
        invalid();
    }
}

void addop()
{
    t = getNextToken(f);
    if (strcmp(t.lexeme, "+") == 0)
        return;
    else if (strcmp(t.lexeme, "-") == 0)
        return;
    else
    {
        printf("Expected + or -\n");
        invalid();
    }
}

void mulop()
{
    t = getNextToken(f);
    if (strcmp(t.lexeme, "*") == 0)
        return;
    else if (strcmp(t.lexeme, "/") == 0)
        return;
    else if (strcmp(t.lexeme, "%") == 0)
        return;
    else
    {
        printf("Expected *, / or %%\n");
        invalid();
    }
}

int main()
{
    f = fopen("input.c", "r");

    if (f == NULL)
    {
        printf("Cannot open file\n");
        exit(0);
    }

    Program();
```

}

---

sample.c (correct)

```c
int main() {
    int a, b[50], c;
    a = b * c;

    a = 10;
    a = b * c;
    g = h+ k;
    m = a >= 9;

    if(a<=9) {
      sum = 10;
    } else {
      sum = 15;
    }

    while(a < 5) {
      a = a+2;
    }

    for(g = 7; g<10;g=g+1) {
      g = g-1;
    }
}
```

OUTPUT :

```
student@lplab-ThinkCentre-M71e: ~/190905494/CD/Week 8
student@lplab-ThinkCentre-M71e:~/190905494/CD/Week 8$ gcc parser.c -o q1; ./q1
Success


=== Tokens Generated are ===
0. <main, 23, 3, keyword>
1. <(, 23, 7, left bracket>
2. <), 23, 8, right bracket>
3. <{, 23, 10, LC>
4. <int, 24, 5, keyword>
5. <a, 24, 9, identifier>
6. <,, 24, 10, special symbol>
7. <b, 24, 12, identifier>
8. <[, 24, 13, LS>
9. <50, 24, 14, number>
10. <], 24, 16, RS>
11. <,, 24, 17, special symbol>
12. <c, 24, 19, identifier>
13. <;, 24, 20, special symbol>
14. <a, 25, 5, identifier>
15. <=, 25, 7, assignment op>
16. <b, 25, 9, identifier>
17. <*, 25, 11, arithmetic operator>
18. <c, 25, 13, identifier>
19. <;, 25, 14, special symbol>
20. <a, 27, 5, identifier>
21. <=, 27, 7, assignment op>
22. <10, 27, 9, number>
23. <;, 27, 11, special symbol>
24. <a, 28, 5, identifier>
25. <=, 28, 7, assignment op>
26. <b, 28, 9, identifier>
27. <*, 28, 11, arithmetic operator>
28. <c, 28, 13, identifier>
29. <;, 28, 14, special symbol>
30. <g, 29, 5, identifier>
31. <=, 29, 7, assignment op>
32. <h, 29, 9, identifier>
33. <+, 29, 10, arithmetic operator>
34. <k, 29, 12, identifier>
35. <;, 29, 13, special symbol>
36. <m, 30, 5, identifier>
```

sample.c (incorrect) with '=' missing

```
parser.c — Week 8    la.h    input.c
int main() {
    int a, b[50], c;
    a = b * c;

    a = 10;
    a = b * c;
    g = h+ k;
    m = a >= 9;

    if(a<=9) {
      sum | 10;
    } else {
      sum = 15;
    }

    while(a < 5) {
      a = a+2;
    }

    for(g = 7; g<10;g=g+1) {
      g = g-1;
    }
}
```

OUTPUT :

```
student@lplab-ThinkCentre-M71e: ~/190905494/CD/Week 8
student@lplab-ThinkCentre-M71e:~/190905494/CD/Week 8$ gcc parser.c -o q1; ./q1
Expected =
at position (row : 11, col : 12)

ERROR
student@lplab-ThinkCentre-M71e:~/190905494/CD/Week 8$
```