

CD LAB WEEK8

Name: Sagnik Chatterjee

Reg:180905478

Roll No :61

Sem: 5

Sec: B

Q1

Lex.c

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <ctype.h>
```

```
#include <string.h>
```

```
const char *keywords[] = {
```

```
    "auto",
```

```
    "double",
```

```
    "int",
```

```
    "struct",
```

```
    "break",
```

```
    "else",
```

```
    "long",
```

```
    "switch",
```

```
    "case",
```

```
    "enum",
```

"register",

"typedef",

"char",

"extern",

"return",

"union",

"continue",

"for",

"signed",

"void",

"do",

"if",

"static",

"while",

"default",

"goto",

"sizeof",

"volatile",

"const",

"float",

"short",

"unsigned",

"printf",

"scanf",

"true", "false"

```
};  
const char *datypes[] = {"int", "char", "void", "float", "bool"};  
int isdtype(char *w)  
{  
    int i;  
    for (i = 0; i < sizeof(datypes) / sizeof(char*); i++)  
    {  
        if (strcmp(w, datypes[i]) == 0)  
        {  
            return 1;  
        }  
    }  
    return 0;  
}  
int isKeyword(char *w)  
{  
  
    int i;  
  
    for (i = 0; i < sizeof(keywords) / sizeof(char*); i++)  
  
    {  
  
        if (strcmp(w, keywords[i]) == 0)
```

```

        {

            return 1;

        }

    }

    return 0;
}
struct token
{
    char lexeme[128];
    unsigned int row, col;
    char type[64];
};
struct sttable
{
    int sno;
    char lexeme[128];
    char dtype[64];
    char type[64];
    int size;
};
int findTable(struct sttable *tab, char *nam, int n)
{
    int i = 0;
    for (i = 0; i < n; i++)
    {
        if (strcmp(tab[i].lexeme, nam) == 0)
        {
            return 1;
        }
    }
    return 0;
}
struct sttable fillTable(int sno, char *lexn, char *dt, char *t, int s)

```

```

{
    struct sttable tab;
    tab.sno = sno;
    strcpy(tab.lexeme, lexn);
    strcpy(tab.dtype, dt);
    strcpy(tab.type, t);
    tab.size = s;
    return tab;
}

void printTable(struct sttable *tab, int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d %s %s %s %d\n", tab[i].sno, tab[i].lexeme, tab[i].dtype,
tab[i].type, tab[i].size);
    }
}

static int row = 1, col = 1;
char buf[2048];
char dbuf[128];
int ind = 0;
const char specialsymbols[] = {'?', '!', ':', ','};
const char arithmeticsymbols[] = {'*'};
int charls(int c, const char *arr)
{
    int len;
    if (arr == specialsymbols)
    {
        len = sizeof(specialsymbols) / sizeof(char);
    }
    else if (arr == arithmeticsymbols)
    {
        len = sizeof(arithmeticsymbols) / sizeof(char);
    }
    for (int i = 0; i < len; i++)
    {
        if (c == arr[i])

```

```

        {
            return 1;
        }
    }
    return 0;
}

void fillToken(struct token *tkn, char c, int row, int col, char *type)
{
    tkn->row = row;
    tkn->col = col;
    strcpy(tkn->type, type);
    tkn->lexeme[0] = c;
    tkn->lexeme[1] = '\0';
}

void newLine()
{
    ++row;
    col = 1;
}

int sz(char *w)
{
    if (strcmp(w, "int") == 0)
        return 4;
    if (strcmp(w, "char") == 0)
        return 1;
    if (strcmp(w, "void") == 0)
        return 0;
    if (strcmp(w, "float") == 0)
        return 8;
    if (strcmp(w, "bool") == 0)
        return 1;
}

struct token getNextToken(FILE *fa)
{
    int c;
    struct token tkn =
    {

```

```

        .row = -1
};
int gotToken = 0;
while (!gotToken && (c = fgetc(fa)) != EOF)
{
    if (charIs(c, specialsymbols))
    {
        fillToken(&tkn, c, row, col, "SS");
        gotToken = 1;
        ++col;
    }
    else if (charIs(c, arithmeticsymbols))
    {
        fseek(fa, -1, SEEK_CUR);
        c = getc(fa);
        if (isalnum(c)) {
            fillToken(&tkn, c, row, col, "ARITHMETICOPERATOR");
            gotToken = 1;
            ++col;
        }
        fseek(fa, 1, SEEK_CUR);
    }
    else if (c == '(')
    {
        fillToken(&tkn, c, row, col, "LB");
        gotToken = 1;
        col++;
    }
    else if (c == ')')
    {
        fillToken(&tkn, c, row, col, "RB");
        gotToken = 1;
        col++;
    }
    else if (c == '{')
    {
        fillToken(&tkn, c, row, col, "LC");

```

```

        gotToken = 1;
        col++;
    }
    else if (c == '}')
    {
        fillToken(&tkn, c, row, col, "RC");
        gotToken = 1;
        col++;
    }
    else if (c == '[')
    {
        fillToken(&tkn, c, row, col, "LS");
        gotToken = 1;
        col++;
    }
    else if (c == ']')
    {
        fillToken(&tkn, c, row, col, "RS");
        gotToken = 1;
        col++;
    }
    else if (c == '+')
    {
        int x = fgetc(fa);
        if (x != '+')
        {
            fillToken(&tkn, c, row, col, "ARITHMETICOPERATOR");
            gotToken = 1;
            col++;
            fseek(fa, -1, SEEK_CUR);
        }
        else
        {
            fillToken(&tkn, c, row, col, "UNARYOPERATOR");
            strcpy(tkn.lexeme, "++");
            gotToken = 1;
            col += 2;
        }
    }

```



```

    }
}
else if (c == '-')
{
    int x = fgetc(fa);
    if (x != '-')
    {
        fillToken(&tkn, c, row, col, "ARITHMETICOPERATOR");
        gotToken = 1;
        col++;
        fseek(fa, -1, SEEK_CUR);
    }
    else
    {
        fillToken(&tkn, c, row, col, "UNARYOPERATOR");
        strcpy(tkn.lexeme, "--");
        gotToken = 1;
        col += 2;
    }
}
else if (c == '=')
{
    int x = fgetc(fa);
    if (x != '=')
    {
        fillToken(&tkn, c, row, col, "ASSIGNMENTOPERATOR");
        gotToken = 1;
        col++;
        fseek(fa, -1, SEEK_CUR);
    }
    else
    {
        fillToken(&tkn, c, row, col, "RELATIONALOPERATOR");
        strcpy(tkn.lexeme, "++");
        gotToken = 1;
        col += 2;
    }
}

```

```

}
else if (isdigit(c))
{
    fillToken(&tkn, c, row, col++, "NUMBER");
    int j = 1;
    while ((c = fgetc(fa)) != EOF && isdigit(c))
    {
        tkn.lexeme[j++] = c;
        col++;
    }
    tkn.lexeme[j] = '\0';
    gotToken = 1;
    fseek(fa, -1, SEEK_CUR);
}
else if (c == '#')
{
    while ((c = fgetc(fa)) != EOF && c != '\n');
    newLine();
}
else if (c == '\n')
{
    newLine();
    c = fgetc(fa);
    if (c == '#')
    {
        while ((c = fgetc(fa)) != EOF && c != '\n');
        newLine();
    }
    else if (c != EOF)
    {
        fseek(fa, -1, SEEK_CUR);
    }
}
else if (isspace(c))
{
    ++col;
}

```

```

else if (isalpha(c) || c == '_')
{
    tkn.row = row;
    tkn.col = col++;
    tkn.lexeme[0] = c;
    int j = 1;
    while ((c = fgetc(fa)) != EOF && isalnum(c))
    {
        tkn.lexeme[j++] = c;
        col++;
    }
    tkn.lexeme[j] = '\0';
    if (isKeyword(tkn.lexeme))
    {
        strcpy(tkn.type, "KEYWORD");
    }
    else
    {
        strcpy(tkn.type, "IDENTIFIER");
    }
    gotToken = 1;
    fseek(fa, -1, SEEK_CUR);
}
else if (c == '/')
{
    int d = fgetc(fa);
    ++col;
    if (d == '/')
    {
        while ((c = fgetc(fa)) != EOF && c != '\n')
        {
            ++col;
        }
        if (c == '\n')
        {
            newLine();
        }
    }
}

```

```

    }
    else if (d == '*')
    {
        do
        {
            if (d == '\n')
            {
                newLine();
            }
            while ((c == fgetc(fa)) != EOF && c != '*')
            {
                ++col;
                if (c == '\n')
                {
                    newLine();
                }
            }
            ++col;
        } while ((d == fgetc(fa)) != EOF && d != '/' && (++col));
        ++col;
    }
    else
    {
        fillToken(&tkn, c, row, --col, "ARITHMETIC OPERATOR");
        gotToken = 1;
        fseek(fa, -1, SEEK_CUR);
    }
}
else if (c == "")
{
    tkn.row = row;
    tkn.col = col;
    strcpy(tkn.type, "STRING LITERAL");
    int k = 1;
    tkn.lexeme[0] = "";
    while ((c = fgetc(fa)) != EOF && c != "")
    {

```

```

        tkn.lexeme[k++] = c;
        ++col;
    }
    tkn.lexeme[k] = '\0';
    gotToken = 1;
}
else if (c == '<' || c == '>' || c == '!')
{
    fillToken(&tkn, c, row, col, "RELATIONALOPERATOR");
    ++col;
    int d = fgetc(fa);
    if (d == '=')
    {
        ++col;
        strcat(tkn.lexeme, "=");
    }
    else
    {
        if (c == '!')
        {
            strcpy(tkn.type, "LOGICALOPERATOR");
        }
        fseek(fa, -1, SEEK_CUR);
    }
    gotToken = 1;
}
else if (c == '&' || c == '|')
{
    int d = fgetc(fa);
    if (c == d)
    {
        tkn.lexeme[0] = tkn.lexeme[1] = c;
        tkn.lexeme[2] = '\0';
        tkn.row = row;
        tkn.col = col;
        ++col;
        gotToken = 1;
    }
}

```

```

        strcpy(tkn.type, "LOGICALOPERATOR");
    }
    else
    {
        fseek(fa, -1, SEEK_CUR);
    }
    ++col;
}
else
{
    ++col;
}
}
return tkn;
}

```

Main runner code:

```

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include "lex.c"
void program();
void declarations();
void datatype();
void idlist();
void idlistprime();
void assignstat();
void statementlist();
void statement();
void expn();
void eprime();
void simpleexp();
void seprime();
void term();
void tprime();
void factor();

```

```

void decisionstat();
void dprime();
void loopingstat();
void relop();
void addop();
void mulop();
struct token tkn;
FILE *f1;
char *rel[] = {"==", "!=", "<=", ">=", ">", "<"};
char *add[] = {"+", "-"};
char *mul[] = {"*", "/", "%"};
int isrel(char *w)
{
    int i;
    for (i = 0; i < sizeof(rel) / sizeof(char*); i++)
    {
        if (strcmp(w, rel[i]) == 0)
        {
            return 1;
        }
    }
    return 0;
}
int isadd(char *w)
{
    int i;
    for (i = 0; i < sizeof(add) / sizeof(char*); i++)
    {
        if (strcmp(w, add[i]) == 0)
        {
            return 1;
        }
    }
    return 0;
}
int ismul(char *w)
{
    int i;

```

```

for (i = 0; i < sizeof(mul) / sizeof(char*); i++)
{
    if (strcmp(w, mul[i]) == 0)
    {
        return 1;
    }
}
return 0;
}

```

```

int main()
{
    FILE *fa, *fb;
    int ca, cb;
    fa = fopen("in.c", "r");
    if (fa == NULL) {
        printf("Cannot open file \n");
        exit(0);
    }
    fb = fopen("./output.c", "w+");
    ca = getc(fa);
    while (ca != EOF) {
        if (ca == ' ')
        {
            putc(ca, fb);
            while (ca == ' ')
                ca = getc(fa);
        }
        if (ca == '/')
        {
            cb = getc(fa);
            if (cb == '/')
            {

```



```

        while (ca != '\n')
            ca = getc(fa);
    }
    else if (cb == '*')
    {
        do
        {
            while (ca != '*')
                ca = getc(fa);
            ca = getc(fa);
        } while (ca != '/');
    }
    else {
        putc(ca, fb);
        putc(cb, fb);
    }
}
else putc(ca, fb);
ca = getc(fa);
}
fclose(fa);
fclose(fb);
fa = fopen("./output.c", "r");
if (fa == NULL) {
    printf("Cannot open file");
    return 0;
}
fb = fopen("temp.c", "w+");
ca = getc(fa);
while (ca != EOF)
{
    if (ca == "")
    {
        putc(ca, fb);
        ca = getc(fa);
        while (ca != "")
        {

```

```

                putc(ca, fb);
                ca = getc(fa);
            }
        }
        else if (ca == '#')
        {
            while (ca != '\n')
            {
                ca = getc(fa);
            }
            ca = getc(fa);
        }
        putc(ca, fb);
        ca = getc(fa);
    }
    fclose(fa);
    fclose(fb);
    fa = fopen("temp.c", "r");
    fb = fopen("./output.c", "w");
    ca = getc(fa);
    while (ca != EOF) {
        putc(ca, fb); ca = getc(fa);
    }
    fclose(fa);
    fclose(fb);
    remove("temp.c");
    f1 = fopen("./output.c", "r");
    if (f1 == NULL)
    {
        printf("Error! File cannot be opened!\n");
        return 0;
    }
    while ((tkn = getNextToken(f1)).row != -1)
    {
        if (strcmp(tkn.lexeme, "main") == 0)
        {
            program();

```

```

        break;
    }
}
printf("Compiled sucessfully\n");
fclose(f1);
}

```

```

void program()
{
    if (strcmp(tkn.lexeme, "main") == 0)
    {
        tkn = getNextToken(f1);
        if (strcmp(tkn.lexeme, "(") == 0)
        {
            tkn = getNextToken(f1);
            if (strcmp(tkn.lexeme, ")") == 0)
            {
                tkn = getNextToken(f1);
                if (strcmp(tkn.lexeme, "{") == 0)
                {
                    tkn = getNextToken(f1);
                    declarations();
                    statementlist();
                    if (strcmp(tkn.lexeme, "}") == 0)
                    {
                        return;
                    }
                }
                else if (strcmp(tkn.lexeme, "for") == 0 ||
                        strcmp("while", tkn.lexeme) == 0)
                {
                    loopingstat();
                    if (strcmp(tkn.lexeme, "}") == 0)
                    {

```

```

        return; exit(0);
    }
    else if (strcmp(tkn.lexeme, "for") == 0 ||
             strcmp("while", tkn.lexeme) == 0)
    {
        loopingstat();
    }
    else if (strcmp(tkn.lexeme, "if") == 0)
    {
        decisionstat();
    }
    else
    {
        printf("{} missing at row=%d col=%d",
tkn.row, tkn.col);
        exit(1);
    }
}
else if (strcmp(tkn.lexeme, "if") == 0)
{
    decisionstat();
    if (strcmp(tkn.lexeme, "{") == 0)
    {
        return;
    }
    else if (strcmp(tkn.lexeme, "for") == 0 ||
             strcmp("while", tkn.lexeme) == 0)
    {
        loopingstat();
    }
    else if (strcmp(tkn.lexeme, "if") == 0)
    {
        decisionstat();
    }
    else
    {

```

```

                                printf("{} missing at row=%d col=%d",
tkn.row, tkn.col);
                                exit(1);
                                }
                                }
                                else
                                {
                                printf("{} missing at row=%d col=%d", tkn.row,
tkn.col);
                                exit(1);
                                }
                                }
                                else
                                {
                                printf("{} missing at row=%d col=%d", tkn.row, tkn.col);
                                exit(1);
                                }
                                }
                                else {
                                printf("{} missing at row=%d col=%d", tkn.row, tkn.col);
                                exit(1);
                                }
                                }
                                else
                                {
                                printf("( missing at row=%d col=%d", tkn.row, tkn.col);
                                exit(1);
                                }
                                }
                                }
void declarations()
{
    if (isdtype(tkn.lexeme) == 0)
    {
        return;
    }
    datatype();

```

```

idlist();
if (strcmp(tkn.lexeme, ";") == 0)
{
    tkn = getNextToken(f1);
    declarations();
}
else
{
    printf("; missing at row=%d col=%d", tkn.row, tkn.col);
    exit(1);
}
}
void datatype()
{
    if (strcmp(tkn.lexeme, "int") == 0)
    {
        tkn = getNextToken(f1);
        return;
    }
    else if (strcmp(tkn.lexeme, "char") == 0)
    {
        tkn = getNextToken(f1);
        return;
    }
    else
    {
        printf("%s Missing datatype at row=%d col=%d", tkn.lexeme, tkn.row,
tkn.col);
        exit(1);
    }
}
void idlist()
{
    if (strcmp(tkn.type, "IDENTIFIER") == 0) {
        tkn = getNextToken(f1);
        idlistprime();
    }
}

```

```

else
{
    printf("Missing IDENTIFIER at row=%d col=%d", tkn.row, tkn.col);
    exit(1);
}
}
void idlistprime()
{
    if (strcmp(tkn.lexeme, ",") == 0)
    {
        tkn = getNextToken(f1);
        idlist();
    }
    if (strcmp(tkn.lexeme, "[") == 0)
    {
        tkn = getNextToken(f1);
        if (strcmp(tkn.type, "NUMBER") == 0)
        {
            tkn = getNextToken(f1);
            if (strcmp(tkn.lexeme, "]") == 0)
            {
                tkn = getNextToken(f1);
                if (strcmp(tkn.lexeme, ",") == 0)
                {
                    tkn = getNextToken(f1);
                    idlist();
                }
                else
                {
                    return;
                }
            }
        }
        else
        {
            printf("] missing at row=%d col=%d", tkn.row, tkn.col);
            exit(1);
        }
    }
}

```

```

    }
}
else
{
    return;
}
}
void statementlist()
{
    if (strcmp(tkn.type, "IDENTIFIER") != 0)
    {
        return;
    }
    statement();
    statementlist();
}
void statement()
{
    if (strcmp(tkn.type, "IDENTIFIER") == 0)
    {
        assignstat();
        if (strcmp(tkn.lexeme, ";") == 0)
        {
            tkn = getNextToken(f1);
            return;
        }
        else
        {
            printf("; missing at row=%d col=%d", tkn.row, tkn.col);
            exit(1);
        }
    }
    if (strcmp(tkn.lexeme, "if") == 0)
    {
        decisionstat();
    }
    if (strcmp(tkn.lexeme, "while") == 0 || strcmp(tkn.lexeme, "for") == 0)
    {

```



```

        loopingstat();
    }
}
void assignstat()
{
    if (strcmp(tkn.type, "IDENTIFIER") == 0)
    {
        tkn = getNextToken(f1);
        if (strcmp(tkn.lexeme, "=") == 0)
        {
            tkn = getNextToken(f1);
            expn();
        }
        else
        {
            printf("= missing at row=%d col=%d", tkn.row, tkn.col);
            exit(1);
        }
    }
    else
    {
        printf("Missing IDENTIFIER at row=%d col=%d", tkn.row, tkn.col);
        exit(1);
    }
} void expn()
{
    simpleexp();
    eprime();
}
void eprime()
{
    if (isrel(tkn.lexeme) == 0)
    {
        return;
    }
    relop();
    simpleexp();
}

```

```

}
void simpleexp()
{
    term();
    seprime();
}
void seprime()
{
    if (isadd(tkn.lexeme) == 0)
    {
        return;
    }
    addop();
    term();
    seprime();
}
void term()
{
    factor();
    tprime();
}
void tprime()
{
    if (ismul(tkn.lexeme) == 0)
    {
        return;
    }
    mulop();
    factor();
    tprime();
}
void factor()
{
    if (strcmp(tkn.type, "IDENTIFIER") == 0)
    {
        tkn = getNextToken(f1);
        return;
    }

```

```

    }
    else if (strcmp(tkn.type, "NUMBER") == 0) {
        tkn = getNextToken(f1);
        return;
    }
}
void decisionstat()
{
    if (strcmp(tkn.lexeme, "if") == 0)
    {
        tkn = getNextToken(f1);
        if (strcmp(tkn.lexeme, "(") == 0)
        {
            tkn = getNextToken(f1);
            expn();
            if (strcmp(tkn.lexeme, ")") == 0)
            {
                tkn = getNextToken(f1);
                if (strcmp(tkn.lexeme, "{") == 0)
                {
                    tkn = getNextToken(f1);
                    statementlist();
                    if (strcmp(tkn.lexeme, "}") == 0)
                    {
                        tkn = getNextToken(f1);
                        dprime();
                    }
                    else
                    {
                        printf("{} missing at row=%d col=%d", tkn.row,
tkn.col);
                        exit(1);
                    }
                }
            }
            else
            {
                printf("{} missing at row=%d col=%d", tkn.row, tkn.col);

```

```

        exit(1);
    }
}
else
{
    printf("") missing at row=%d col=%d", tkn.row, tkn.col);
    exit(1);
}
}
else
{
    printf("( missing at row=%d col=%d", tkn.row, tkn.col);
    exit(1);
}
}
}
void dprime() {
    if (strcmp(tkn.lexeme, "else") == 0)
    {
        tkn = getNextToken(f1);
        if (strcmp(tkn.lexeme, "{") == 0)
        {
            tkn = getNextToken(f1);
            statementlist();
            if (strcmp(tkn.lexeme, ";") == 0)
            {
                tkn = getNextToken(f1);
                return;
            }
            else
            {
                printf("{} missing at row=%d col=%d", tkn.row, tkn.col);
                exit(1);
            }
        }
        else
        {

```

```

        printf("{ missing at row=%d col=%d", tkn.row, tkn.col);
        exit(1);
    }
}
else
{
    return;
}
}
void loopingstat()
{
    if (strcmp(tkn.lexeme, "while") == 0)
    {
        tkn = getNextToken(f1);
        if (strcmp(tkn.lexeme, "(") == 0)
        {
            tkn = getNextToken(f1);
            expn();
            if (strcmp(tkn.lexeme, ")") == 0)
            {
                tkn = getNextToken(f1);
                if (strcmp(tkn.lexeme, "{") == 0)
                {
                    tkn = getNextToken(f1);
                    statementlist();
                    if (strcmp(tkn.lexeme, "}") == 0)
                    {
                        tkn = getNextToken(f1);
                        return;
                    }
                }
                else {
                    printf("{ missing at row=%d col=%d", tkn.row,
tkn.col);
                    exit(1);
                }
            }
        }
    }
    else

```

```

        {
            printf("{ missing at row=%d col=%d", tkn.row, tkn.col);
            exit(1);
        }
    }
    else
    {
        printf(") missing at row=%d col=%d", tkn.row, tkn.col);
        exit(1);
    }
}
else
{
    printf("(" missing at row=%d col=%d", tkn.row, tkn.col);
    exit(1);
}
}
else if (strcmp(tkn.lexeme, "for") == 0)
{
    tkn = getNextToken(f1);
    if (strcmp(tkn.lexeme, "(") == 0)
    {
        tkn = getNextToken(f1);
        assignstat();
        if (strcmp(tkn.lexeme, ";") == 0)
        {
            tkn = getNextToken(f1);
            expn();
            if (strcmp(tkn.lexeme, ";") == 0)
            {
                tkn = getNextToken(f1);
                assignstat();
                if (strcmp(tkn.lexeme, ")") == 0)
                {
                    tkn = getNextToken(f1);
                    if (strcmp(tkn.lexeme, "{") == 0)
                    {

```

```

tkn = getNextToken(f1);
statementlist();
if (strcmp(tkn.lexeme, "{") == 0)
{
    tkn = getNextToken(f1);
    return;
}
else
{
    printf(" } missing at row=%d col=%d",
tkn.row, tkn.col);

    exit(1);
}
}
else
{
    printf("{ missing at row=%d col=%d ",
tkn.row, tkn.col);

    exit(1);
}
}
else
{
    printf(") missing at row=%d col=%d", tkn.row,
tkn.col);

    exit(1);
}
}
else
{
    printf("; missing at row=%d col=%d", tkn.row, tkn.col);
    exit(1);
}
}
else
{
    printf("; missing at row=%d col=%d", tkn.row, tkn.col);

```

```

        exit(1);
    }
}
else
{
    printf("( missing at row=%d col=%d", tkn.row, tkn.col);
    exit(1);
}
}
}
void relop()
{
    if (strcmp(tkn.lexeme, "==") == 0)
    {
        tkn = getNextToken(f1);
        return;
    }
    if (strcmp(tkn.lexeme, "!=") == 0)
    {
        tkn = getNextToken(f1);
        return;
    }
    if (strcmp(tkn.lexeme, "<=") == 0)
    {
        tkn = getNextToken(f1); return;
    }
    if (strcmp(tkn.lexeme, ">=") == 0)
    {
        tkn = getNextToken(f1);
        return;
    }
    if (strcmp(tkn.lexeme, "<") == 0)
    {
        tkn = getNextToken(f1);
        return;
    }
    if (strcmp(tkn.lexeme, ">") == 0)

```

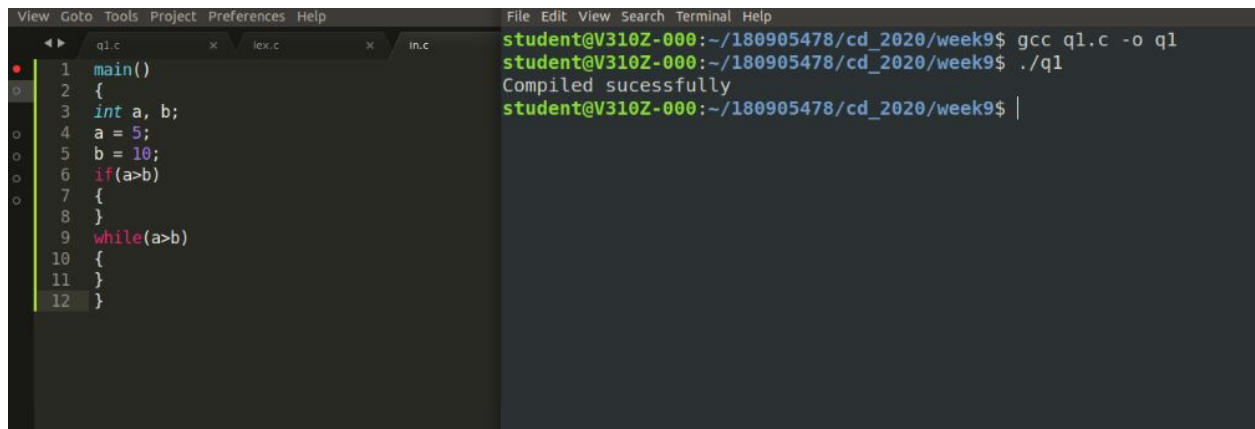


```

    {
        tkn = getNextToken(f1);
        return;
    }
}
void addop()
{
    if (strcmp(tkn.lexeme, "+") == 0)
    {
        tkn = getNextToken(f1);
        return;
    }
    if (strcmp(tkn.lexeme, "-") == 0)
    {
        tkn = getNextToken(f1);
        return;
    }
}
void mulop()
{
    if (strcmp(tkn.lexeme, "*") == 0)
    {
        tkn = getNextToken(f1);
        return;
    }
    if (strcmp(tkn.lexeme, "/") == 0)
    {
        tkn = getNextToken(f1);
        return;
    }
    if (strcmp(tkn.lexeme, "%") == 0)
    {
        tkn = getNextToken(f1);
        return;
    }
}
}

```

## Screenshot:

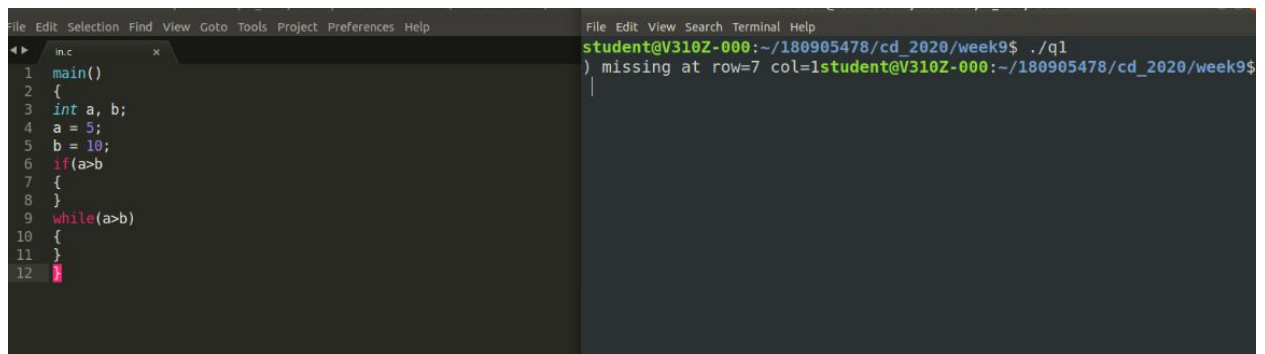


The screenshot shows a code editor with three tabs: q1.c, lex.c, and in.c. The q1.c tab is active, displaying the following C code:

```
1 main()
2 {
3     int a, b;
4     a = 5;
5     b = 10;
6     if(a>b)
7     {
8     }
9     while(a>b)
10    {
11    }
12 }
```

The terminal on the right shows the following commands and output:

```
student@V310Z-000:~/180905478/cd_2020/week9$ gcc q1.c -o q1
student@V310Z-000:~/180905478/cd_2020/week9$ ./q1
Compiled successfully
student@V310Z-000:~/180905478/cd_2020/week9$ |
```



The screenshot shows the same code editor and terminal as the previous image. The code in q1.c is the same. The terminal shows the command `./q1` being executed, but it results in an error:

```
student@V310Z-000:~/180905478/cd_2020/week9$ ./q1
) missing at row=7 col=1student@V310Z-000:~/180905478/cd_2020/week9$ |
```

The error message indicates a syntax error in the code, specifically a missing closing parenthesis at row 7, column 1. This corresponds to the opening curly brace of the `if(a>b)` block in the code.