#include <stdio.h>

#include <ctype.h>

int charClass;

char lexeme[100];

char nextChar;

int lexLen;

int token;

int nextToken;

FILE\* in\_fp, \* fopen();

void addChar();

void getChar();

void getNonBlank();

int lex();

#define LETTER 0

#define DIGIT 1

#define UNKNOWN 99

#define INT\_LIT 10

#define IDENT 11

#define ASSIGN\_OP 20

#define ADD\_OP 21

#define SUB\_OP 22

#define MULT\_OP 23

#define DIV\_OP 24

#define LEFT\_PAREN 25

#define RIGHT\_PAREN 26

main() {

if ((in\_fp = fopen("front.in", "r")) == NULL)

printf("ERROR - cannot open front.in \n");

else {

getChar();

do {

lex();

} while (nextToken!= EOF);

}

}

int lookup(char ch) {

switch (ch) {

case '(':

addChar();

nextToken = LEFT\_PAREN;

break;

case ')':

addChar();

nextToken = RIGHT\_PAREN;

break;

case '+':

addChar();

nextToken = ADD\_OP;

break;

case '-':

addChar();

nextToken = SUB\_OP;

break;

case '\*':

addChar();

nextToken = MULT\_OP;

break;

case '/':

addChar();

nextToken = DIV\_OP;

break;

default:

addChar();

nextToken = EOF;

break;

}

return nextToken;

}

void addChar() {

if (lexLen <= 98) {

lexeme[lexLen++] = nextChar;

lexeme[lexLen] = 0;

}

else

printf("Error - lexeme is too long \n");

}

void getChar() {

if ((nextChar = getc(in\_fp)) = EOF) {

if (isalpha(nextChar))

charClass = LETTER;

else if (isdigit(nextChar))

charClass = DIGIT;

else charClass = UNKNOWN;

}

else

charClass = EOF;

}

void getNonBlank() {

while (isspace(nextChar))

getChar();

}

int lex() {

lexLen = 0;

getNonBlank();

switch (charClass) {

case LETTER:

addChar();

getChar();

while (charClass == LETTER || charClass == DIGIT) {

addChar();

getChar();

}

nextToken = IDENT;

break;

case DIGIT:

addChar();

getChar();

while (charClass == DIGIT) {

addChar();

getChar();

}

nextToken = INT\_LIT;

break;

case UNKNOWN:

lookup(nextChar);

getChar();

break;

case EOF:

nextToken = EOF;

lexeme[0] = 'E';

lexeme[1] = 'O';

lexeme[2] = 'F';

lexeme[3] = 0;

break;

}

printf("Next token is: %d, Next lexeme is %s\n",

nextToken, lexeme);

return nextToken;

}

---

#include <stdio.h>

- Includes the standard input/output library for functions like printf and getc.

#include <ctype.h>

- Includes the character handling library for functions like isalpha and isdigit.

int charClass;

- Declares an integer to store the class of the current character (LETTER, DIGIT, UNKNOWN, or EOF).

char lexeme[100];

- Declares a char array of size 100 to hold the current lexeme (token string).

char nextChar;

- Declares a char to store the next character read from the file.

int lexLen;

- Declares an integer to track the length of the current lexeme.

int token;

- Declares an integer for the current token (unused in this code).

int nextToken;

- Declares an integer to store the type of the next token (e.g., IDENT, INT\_LIT).

FILE\* in\_fp, \* fopen();

- Declares a file pointer for reading the input file and mentions fopen (redundant since it’s in stdio.h).

void addChar();

- Declares a function to add a character to the lexeme.

void getChar();

- Declares a function to read and classify the next character.

void getNonBlank();

- Declares a function to skip whitespace characters.

int lex();

- Declares a function to perform lexical analysis and return a token.

#define LETTER 0

- Defines a constant for alphabetic characters as 0.

#define DIGIT 1

- Defines a constant for numeric digits as 1.

#define UNKNOWN 99

- Defines a constant for symbols or unrecognized characters as 99.

#define INT\_LIT 10

- Defines a constant for integer literals (e.g., 123) as 10.

#define IDENT 11

- Defines a constant for identifiers (e.g., abc123) as 11.

#define ASSIGN\_OP 20

- Defines a constant for the assignment operator (=) as 20 (unused here).

#define ADD\_OP 21

- Defines a constant for the addition operator (+) as 21.

#define SUB\_OP 22

- Defines a constant for the subtraction operator (-) as 22.

#define MULT\_OP 23

- Defines a constant for the multiplication operator (\*) as 23.

#define DIV\_OP 24

- Defines a constant for the division operator (/) as 24.

#define LEFT\_PAREN 25

- Defines a constant for the left parenthesis (() as 25.

#define RIGHT\_PAREN 26

- Defines a constant for the right parenthesis ()) as 26.

main() {

- Starts the main function (implicitly returns int in older C standards).

if ((in\_fp = fopen("front.in", "r")) == NULL)

- Opens front.in in read mode and checks if it fails (returns NULL).

printf("ERROR - cannot open front.in \n");

- Prints an error message if the file can’t be opened.

else {

- Starts the else block if the file opens successfully.

getChar();

- Reads the first character from the file.

do {

- Starts a do-while loop to process tokens.

lex();

- Calls the lex() function to analyze and print the next token.

} while (nextToken!= EOF);

- Continues the loop until the token is EOF (end of file).

}

- Closes the else block.

}

- Closes the main function (no explicit return or file close).

int lookup(char ch) {

- Starts a function that takes a character and returns its token type.

switch (ch) {

- Switches based on the input character.

case '(':

- Checks if the character is (.

addChar();

- Adds ( to the lexeme.

nextToken = LEFT\_PAREN;

- Sets the token to LEFT\_PAREN (25).

break;

- Exits the switch.

case ')':

- Checks if the character is ).

addChar();

- Adds ) to the lexeme.

nextToken = RIGHT\_PAREN;

- Sets the token to RIGHT\_PAREN (26).

break;

- Exits the switch.

case '+':

- Checks if the character is +.

addChar();

- Adds + to the lexeme.

nextToken = ADD\_OP;

- Sets the token to ADD\_OP (21).

break;

- Exits the switch.

case '-':

- Checks if the character is -.

addChar();

- Adds - to the lexeme.

nextToken = SUB\_OP;

- Sets the token to SUB\_OP (22).

break;

- Exits the switch.

case '\*':

- Checks if the character is \*.

addChar();

- Adds \* to the lexeme.

nextToken = MULT\_OP;

- Sets the token to MULT\_OP (23).

break;

- Exits the switch.

case '/':

- Checks if the character is /.

addChar();

- Adds / to the lexeme.

nextToken = DIV\_OP;

- Sets the token to DIV\_OP (24).

break;

- Exits the switch.

default:

- Handles any character not listed above.

addChar();

- Adds the character to the lexeme.

nextToken = EOF;

- Sets the token to EOF (-1), likely unintended for all unknown characters.

break;

- Exits the switch.

}

- Closes the switch block.

return nextToken;

- Returns the determined token.

}

- Closes the lookup function.

void addChar() {

- Starts a function to add a character to the lexeme.

if (lexLen <= 98) {

- Checks if the lexeme length is 98 or less (leaves room for null terminator).

lexeme[lexLen++] = nextChar;

- Adds the current character to lexeme and increments lexLen.

lexeme[lexLen] = 0;

- Adds a null terminator to make lexeme a valid C string.

} else

- Starts the else block if the lexeme is too long.

printf("Error - lexeme is too long \n");

- Prints an error message if lexeme exceeds 98 characters.

}

- Closes the addChar function.

void getChar() {

- Starts a function to read and classify the next character.

if ((nextChar = getc(in\_fp)) = EOF) {

- Reads a character from the file (bug: uses = instead of == for EOF check).

if (isalpha(nextChar))

- Checks if the character is alphabetic.

charClass = LETTER;

- Sets charClass to LETTER (0).

else if (isdigit(nextChar))

- Checks if the character is a digit.

charClass = DIGIT;

- Sets charClass to DIGIT (1).

else charClass = UNKNOWN;

- Sets charClass to UNKNOWN (99) for other characters.

} else

- Starts the else block if EOF is encountered (bug reverses logic).

charClass = EOF;

- Sets charClass to EOF (-1).

}

- Closes the getChar function.

void getNonBlank() {

- Starts a function to skip whitespace characters.

while (isspace(nextChar))

- Loops while the current character is whitespace.

getChar();

- Reads the next character.

}

- Closes the getNonBlank function.

int lex() {

- Starts a function to analyze and return tokens.

lexLen = 0;

- Resets the lexeme length to 0.

getNonBlank();

- Skips any leading whitespace.

switch (charClass) {

- Switches based on the character class.

case LETTER:

- Handles alphabetic characters.

addChar();

- Adds the character to the lexeme.

getChar();

- Reads the next character.

while (charClass == LETTER || charClass == DIGIT) {

- Loops while the next character is a letter or digit.

addChar();

- Adds the character to the lexeme.

getChar();

- Reads the next character.

}

- Closes the while loop.

nextToken = IDENT;

- Sets the token to IDENT (11).

break;

- Exits the switch.

case DIGIT:

- Handles numeric digits.

addChar();

- Adds the digit to the lexeme.

getChar();

- Reads the next character.

while (charClass == DIGIT) {

- Loops while the next character is a digit.

addChar();

- Adds the digit to the lexeme.

getChar();

- Reads the next character.

}

- Closes the while loop.

nextToken = INT\_LIT;

- Sets the token to INT\_LIT (10).

break;

- Exits the switch.

case UNKNOWN:

- Handles symbols or operators.

lookup(nextChar);

- Determines the token for the current character.

getChar();

- Reads the next character.

break;

- Exits the switch.

case EOF:

- Handles the end of the file.

nextToken = EOF;

- Sets the token to EOF (-1).

lexeme[0] = 'E';

- Sets the first character of lexeme to E.

lexeme[1] = 'O';

- Sets the second character of lexeme to O.

lexeme[2] = 'F';

- Sets the third character of lexeme to F.

lexeme[3] = 0;

- Sets the fourth character to null terminator for "EOF".

break;

- Exits the switch.

}

- Closes the switch block.

printf("Next token is: %d, Next lexeme is %s\n", nextToken, lexeme);

- Prints the token number and lexeme string.

return nextToken;

- Returns the token.

}

- Closes the lex function.

---