#include <iostream>

#include <fstream>

#include <cctype>

#include <string>

using namespace std;

int charClass;

string lexeme;

char nextChar;

int lexLen;

int token;

int nextToken;

ifstream in\_fp;

void addChar();

void getChar();

void getNonBlank();

int lex();

const int LETTER = 0;

const int DIGIT = 1;

const int UNKNOWN = 99;

const int INT\_LIT = 10;

const int IDENT = 11;

const int ASSIGN\_OP = 20;

const int ADD\_OP = 21;

const int SUB\_OP = 22;

const int MULT\_OP = 23;

const int DIV\_OP = 24;

const int LEFT\_PAREN = 25;

const int RIGHT\_PAREN = 26;

int main() {

in\_fp.open("front.in");

if (!in\_fp.is\_open()) {

cout << "ERROR - cannot open front.in" << endl;

} else {

getChar();

do {

lex();

} while (nextToken != EOF);

}

in\_fp.close();

return 0;

}

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 += nextChar;

lexLen++;

} else {

cout << "Error - lexeme is too long" << endl;

}

}

void getChar() {

if (in\_fp.get(nextChar)) {

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;

lexeme.clear();

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 = "EOF";

break;

}

cout << "Next token is: " << nextToken << ", Next lexeme is " << lexeme << endl;

return nextToken;

}

---

#include <iostream>

- Includes the input/output stream library for using cout to print to the console.

#include <fstream>

- Includes the file stream library for using ifstream to read from files.

#include <cctype>

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

#include <string>

- Includes the string class library to use string objects for lexeme.

using namespace std;

- Allows use of standard library names (e.g., cout, string) without the std:: prefix.

int charClass;

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

string lexeme;

- Declares a string object 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).

ifstream in\_fp;

- Declares an input file stream object for reading from front.in.

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.

const int LETTER = 0;

- Defines a constant for alphabetic characters as 0.

const int DIGIT = 1;

- Defines a constant for numeric digits as 1.

const int UNKNOWN = 99;

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

const int INT\_LIT = 10;

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

const int IDENT = 11;

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

const int ASSIGN\_OP = 20;

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

const int ADD\_OP = 21;

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

const int SUB\_OP = 22;

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

const int MULT\_OP = 23;

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

const int DIV\_OP = 24;

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

const int LEFT\_PAREN = 25;

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

const int RIGHT\_PAREN = 26;

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

int main() {

- Starts the main function, the program’s entry point, returning an integer.

in\_fp.open(front.in);

- Opens the file front.in for reading using the ifstream object.

if (!in\_fp.is\_open()) {

- Checks if the file failed to open.

cout << ERROR - cannot open front.in << endl;

- Prints an error message to the console 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.

in\_fp.close();

- Closes the input file.

return 0;

- Returns 0 to indicate successful program completion.

}

- Closes the main function.

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.

lexeme += nextChar;

- Appends the current character to the lexeme string.

lexLen++;

- Increments the lexeme length.

} else {

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

cout << Error - lexeme is too long << endl;

- Prints an error message to the console.

}

- Closes the if-else block.

}

- Closes the addChar function.

void getChar() {

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

if (in\_fp.get(nextChar)) {

- Reads a character into nextChar from the file; returns true if successful.

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

- Handles any other character.

charClass = UNKNOWN;

- Sets charClass to UNKNOWN (99).

} else {

- Starts the else block if no character is read (EOF).

charClass = EOF;

- Sets charClass to EOF (-1).

}

- Closes the if-else block.

}

- 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 while loop.

}

- Closes the getNonBlank function.

int lex() {

- Starts a function to analyze and return tokens.

lexLen = 0;

- Resets the lexeme length to 0.

lexeme.clear();

- Clears the lexeme string.

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 = EOF;

- Sets the lexeme to EOF.

break;

- Exits the switch.

}

- Closes the switch block.

cout << Next token is: << nextToken << , Next lexeme is << lexeme << endl;

- Prints the token number and lexeme string to the console.

return nextToken;

- Returns the token.

}

- Closes the lex function.

---

This is the clean explanation for each line, with no extra markers or "cpp". Let me know if you need anything else!