# PROGRAMMING

# LANGUAGES

# 

# ASSIGNMENT 1

05130000304 BERÇEM ÇATALKAYA

05130001204 PINAR AKARSU

05130000277 TUĞÇE DÜLGE

SOURCE CODE:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_STRING\_LENGHT 255

#define MAX\_VAR\_LENGHT 32

// 0513-304 Berçem ÇATALKAYA

// 0513-1204 Pınar AKARSU

// 0513-277 Tuğçe DÜLGE

//deneme.for dosyası ile işlemleri gerçekleştirebilirsiniz.

//.for dosyasının sonuna kadar okuduğu için, for dosyasının sonundaki kod satırından sonra enter tuşuna basılması gerekmektedir.

enum token

{

KEYWORD,

IDENTIFIER,

NUMBER,

STRING,

OPERATOR,

PARANTHESIS,

UNKNOWN

};

enum error

{

MAX\_STRING\_LENGTH\_ERROR,

MAX\_VAR\_LENGTH\_ERROR,

UNKNOWN\_SYMBOL\_ERROR

};

char token\_names[7][20] =

{

"KEYWORD","IDENTIFIER","NUMBER",

"STRING","OPERATOR","PARANTHESIS","UNKNOWN"

};

char \*error\_strings[3] =

{

"Max string length exceeded",

"Max variable name length exceeded",

"Unknown character"

};

char tokenCounts[7] = {0};

int line = 1, column = 0, lastColCount = 0;

void doLexicalAnalysis(FILE \*);

int main()

{

FILE \*inputFile;

if((inputFile = fopen("deneme.for","r")) == NULL)

{

printf("File not exist!");

return 0;

}

printf("\n%8s%15s %s\n\n","POSITION","TOKEN","VALUE");

doLexicalAnalysis(inputFile); // it analyzes.

printResult();

fclose(inputFile); //kapat

return 0;

}

int isLetter(char ch) // It controlles letter or others.

{

if( (ch>='a'&&ch<='z') || (ch>='A'&&ch<='Z') || ch=='\_')

return 1;

return 0;

}

int isDigit(char ch) /{ It controlles digit or others

if(ch>='0'&&ch<='9')

return 1;

return 0;

}

int isOperator(char ch) // It controlles operator or others

{

if(ch=='='||ch=='+'||ch=='/'||ch=='-'

||ch=='\*'||ch=='['||ch==']'

||ch=='.'||ch==','||ch=='\*\*'||ch=='('||ch==')')

return 1;

return 0;

}

int isComment(char ch) // It controlles comment lines

{

if(ch=='c' || ch=='C')

return 1;

return 0;

}

int isBooleanOp(char \*word) It controlles boolean operator

{

char \*booleanOps[9] =

{

".TRUE.", ".AND.",".FALSE.",".OR.",".NOT.",".LT.",".GE.",".EQ.",".NE."

};

int i = 0;

for(i = 0; i < 9; i++)

{

if(!strcasecmp(word,booleanOps[i]))

return i+1;

}

return 0;

}

int isKeyword(char \*word)

{

char \*keywords[60] =

{

"integer", "array","real","double precision","complex","logical","character",

"program","write","read","stop","end","int","real","dble","ichar",

"char","if","endif","then","else","elseif","do","continue","while",

"enddo","do while","goto","until","subroutine","call","parameter",

"return","implicit", "entry","common", "close", "assign",

"backspace", "block data", "equivalence", "external", "format", "function",

"pause", "print", "rewind",

"rewrite", "save","sequence","none"

};

int i = 0;

for(i = 0; i < 59; i++)

{

if(!strcasecmp(word,keywords[i]))

return i+1;

}

return 0;

}

int isSpace(char ch) // ASCII 32=Space. Return one. Else return zero.

{

switch(ch)

{

case 9 ... 13:

case 32:

return 1;

}

return 0;

}

int isString(char ch) // It controlles string or others

{

if(ch=='\'')

return 1;

return 0;

}

char fgetcWithCounter(FILE \*file) // it reads a character in file.Then, if character equals a space,column increase.

{

char c = fgetc(file);

if(c == '\n')

{

line++;

lastColCount = column;

column = 0;

}

else

column++;

return c;

}

void fgoBackwards(FILE \*file,int i)

{

fseek(file,-1\*i\*sizeof(char),SEEK\_CUR);

if(column == 0)

{

line--;

column = lastColCount;

}

else if(column > i)

{

column -= i;

}

else

{

line--;

column = lastColCount - (i-column);

}

}

char fgetcWithoutComment(FILE \*file) // In FORTRAN 77, ‘c’ and ‘C’ characters equal to comment line.

{

char p = fgetcWithCounter(file);

if( p != 'c' && p != 'C')

return p;

char c = fgetcWithCounter(file);

if((p == 'c' && c ==32))

{

while(c != '\n')

{

c = fgetcWithCounter(file);

}

return fgetcWithoutComment(file);

}

if((p == 'C' && c ==32))

{

while(c != '\n')

{

c = fgetcWithCounter(file);

}

return fgetcWithoutComment(file);

}

fgoBackwards(file,1);

return p;

}

void printAndWrite(int line, int column, int tokenId, char \*tokenValue)

{

tokenCounts[tokenId]++;

printf("(%2d,%2d) %15s %s \n",line,column,token\_names[tokenId],tokenValue);

}

void printErrorAndWrite(int line, int col, int errorId, char \*value)

{

printf("ERROR: %s at (%2d,%2d) %s\n",error\_strings[errorId],line,col,value);

}

void shortenStringFrom(int i, char \*string)

{

string[i++]='.';

string[i++]='.';

string[i++]='.';

string[i]='\0';

}

void doLexicalAnalysis(FILE \*file) // it reads a character. Then, it controlles is string? is digit? is operatör? is booleans? is unknown? is paranthesis? This function dont read comment lines. So it writes display.

{

int ln,col,i=0;

char c;

char text[MAX\_STRING\_LENGHT];

while((c=fgetcWithoutComment(file)) != EOF)

{

ln = line;

col = column;

if(isLetter(c))

{

i = 0;

while(isLetter(c) || isDigit(c))

{

if(i > MAX\_VAR\_LENGHT)

{

shortenStringFrom(10,text);

printErrorAndWrite(ln,col,MAX\_VAR\_LENGTH\_ERROR,text);

return;

}

text[i++] = c;

c = fgetcWithoutComment(file);

}

text[i] = '\0';

if(isBooleanOp(text))

printAndWrite(ln,col,OPERATOR,text);

else if(isKeyword(text))

printAndWrite(ln,col,KEYWORD,text);

else

printAndWrite(ln,col,IDENTIFIER,text);

fgoBackwards(file,1);

continue;

}

if(isDigit(c))

{

int dot = 0;

i = 0;

while(isDigit(c) || (c == '.' && !dot))

{

if(c=='.')

dot=1;

text[i++] = c;

c = fgetcWithoutComment(file);

}

text[i] = '\0';

printAndWrite(ln,col,NUMBER,text);

fgoBackwards(file,1);

continue;

}

if(isString(c))

{

i = 0;

text[i++] = c;

while(!isString(c = fgetcWithoutComment(file)))

{

if(i > MAX\_STRING\_LENGHT)

{

shortenStringFrom(10,text);

printErrorAndWrite(ln,col,MAX\_STRING\_LENGTH\_ERROR,text);

return;

}

text[i++] = c;

}

text[i++] = c;

text[i] = '\0';

printAndWrite(ln,col,STRING,text);

continue;

}

if(isSpace(c))

continue;

if(isOperator(c))

{

char p = c;

text[0] = p;

text[1] = '\0';

if(p == '\*')

{

c = fgetcWithoutComment(file);

if(c != '\*')

{

printAndWrite(ln,col,OPERATOR,text);

fgoBackwards(file,1);

continue;

}

text[1] = c;

text[2] = '\0';

printAndWrite(ln,col,OPERATOR,text);

continue;

}

else if(p=='.')

{

c = fgetcWithoutComment(file);

if(c != 'L')

{

printAndWrite(ln,col,OPERATOR,text);

fgoBackwards(file,1);

continue;

}

text[1] = c;

text[2] = '\0';

printAndWrite(ln,col,OPERATOR,text);

continue;

}

if(p=='(' ||p==')')

{

printAndWrite(ln,col,PARANTHESIS,text);

continue;

}

else

{

printAndWrite(ln,col,OPERATOR,text);

continue;

}

}

text[0] = c;

text[1] = '\0';

printErrorAndWrite(line,column,UNKNOWN\_SYMBOL\_ERROR,text);

return;

}

}

void printResult() // so, this function writes result.

{

int i = 0;

printf("\nNUMBER OF TOKENS \n");

for(i = 0; i<7 ; i++)

{

printf("%-15s : %d\n",token\_names[i],tokenCounts[i]);

}

}

TOKENS: There are 7 categories. We used enum structure.   
  
enum token

{

KEYWORD,

IDENTIFIER,

NUMBER,

STRING,

OPERATOR,

PARANTHESIS,

UNKNOWN

};

Keyword: 51 keyword and 9 boolean operator unified. The following keywords were defined in [FORTRAN 77](http://fortranwiki.org/fortran/show/FORTRAN+77): assign, backspace, block data, call, close, common, continue, data, dimension, do, else, else if, end, endfile, endif, entry, equivalence, external, format, function, goto, if, implicit, inquire, intrinsic, open, parameter, pause, print, program, read, return, rewind, rewrite, save, stop, subroutine, then, write.

char \*keywords[60] =

{

"integer", "array","real","double precision","complex","logical","character",

"program","write","read","stop","end","int","real","dble","ichar",

"char","if","endif","then","else","elseif","do","continue","while",

"enddo","do while","goto","until","subroutine","call","parameter",

"return","implicit", "entry","common", "close", "assign",

"backspace", "block data", "equivalence", "external", "format", "function",

"pause", "print", "rewind",

"rewrite", "save","sequence","none"

};

char \*booleanOps[9] =

{

".TRUE.", ".AND.",".FALSE.",".OR.",".NOT.",".LT.",".GE.",".EQ.",".NE."

};

The basic arithmetic operations of addition, subtraction, multiplication, division, and exponentiation (raising to a power) are all possible in FORTRAN 77. Addition and subtraction in FORTRAN 77 use the same familiar symbols + and -. However, multiplication (which is denoted in a variety of ways in mathematics) is represented in FORTRAN 77 by an asterisk \* and division by a forward slash /. A double asterisk \*\* is employed to raise a base to a power.  
  
int isOperator(char ch) // operator mu ?

{

if(ch=='='||ch=='+'||ch=='/'||ch=='-'

||ch=='\*'||ch=='['||ch==']'

||ch=='.'||ch==','||ch=='\*\*'||ch=='('||ch==')')

return 1;

return 0;

}

The letters A-Z and digits 0-9 are often grouped together and referred to as alpha-numeric characters. Although this character set seems somewhat limited, it should be available on any computer. The blank character is ignored except within character constants and the currency symbol is not actually used. Of course, most computer keyboards provide many other characters, not least of which are the lower-case letters. Any printable characters may be used within comments and character constants.