15/05/2020

Programming

Languages

Project 1

Lexical Analyzer for Psi++

Oksana Damarja-05170000811

This program accepts a source file called code.psi and produces a text file named code.lex which contains all the tokens of the Psi++ language listed one after the other.

The token types considered, together with the rules followed to analyze them are as followed:

1. **Identifiers.**

* Maximum identifier size is 20 characters. If you use an identifier larger than that, the lexical analyzer issues an error message.
* Psi++ language is not case sensitive and all the identifier names are standardized as upper case.
* Identifiers start with an alphabetic character (a letter) and are composed of one or more letters , digits or\_ (underscore)

1. **Integer Constants.**

* Maximum integer size is 10 digits. If you use an integer value longer than that, the lexical analyzer issues an error message.
* Negative values are not supported.

1. **Operators.**

* Valid operators of the language are +,-,\*,/,++,--,:=

1. **Brackets.**

* LeftPar**: (**
* RightPar**: )**
* LeftSquareBracket: **[**
* RightSquareBracket**: ]**
* LeftCurlyBracket: **{**
* RightCurlyBracket: **}**

1. **String constants.**

* String constants of Psi++ are delimited by double as in “this is a string”
* String constants have unlimited size
* String constants cannot contain the double quote character. when you reach one, the string terminates.
* If a string constant cannot terminate before the file end, there should be a lexical error issued.

1. **Keywords.**

Keywords are: break , case , char , const , continue , do , else , enum , float , for , goto , if , int , long , record , return , static , while

* Psi++ language is not case sensitive and all the keywords are standardized as lower case. You can write the same word as “while” OR “While” OR “WHILE” and they all generate the

1. **End of line.**

* It is represented by ;

1. **Comments.**

* Anything between (\* and \*) is a comment.
* If a comment cannot terminate before the file end, there should be a lexical error issued.
* Comments are just like blank space and they provide no tokens.

Following these rules , I was able to write a program in the C language which read the lines from the code.psi file and analyzed each of the characters. After being analyzed they were written to the code.lex text file , separated in the asked tokens.

The source code of the program is as follows :

Source Code

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

int main()

{

FILE\* filePsi;

FILE\* fileLex;

filePsi = fopen("code.psi.txt", "r"); //read the file code.psi

fileLex = fopen("code.lex.txt", "w"); //write on the file code.lex

int k = 0; //used for the strings array.

int int\_len = 0; //counts the number of integers.

char strings[1000]; //array that keeps the strings read from the file.

int m = 0; //used to print the characters in fileLex.

char words[20]; //keeps the identifiers.

int a;

int b;

int identifier\_length;

int control\_var;

char keywords[18][20]; //the keywords are kept in an array.

strcpy(keywords[0],"break");

strcpy(keywords[1],"case");

strcpy(keywords[2],"char");

strcpy(keywords[3],"const");

strcpy(keywords[4],"continue");

strcpy(keywords[5],"do");

strcpy(keywords[6],"else");

strcpy(keywords[7],"enum");

strcpy(keywords[8],"float");

strcpy(keywords[9],"for");

strcpy(keywords[10],"goto");

strcpy(keywords[11],"if");

strcpy(keywords[12],"int");

strcpy(keywords[13],"long");

strcpy(keywords[14],"record");

strcpy(keywords[15],"return");

strcpy(keywords[16],"static");

strcpy(keywords[17],"while");

char ch;

if(filePsi == NULL){ // if file is not found a message is printed.

printf("File not found!");

}

else

{

ch = fgetc(filePsi);

while(!feof(filePsi)) //reading the file.

{

if(isspace(ch)) //spaces not considered.

{

ch = fgetc(filePsi);

}

//identifiers and keywords control.

if(islower(ch)|| isupper(ch))

{

a = 0;

b = 0;

identifier\_length = 0;

control\_var = 0;

while(isdigit(ch)|| islower(ch)|| isupper(ch)|| (ch == '\_'))

{

if(a > 20 )

{

fputs("Lexical Error.The identifier cannot exceed 20 characters.\n", fileLex);

break;

}

if(islower(ch))

{

toupper(ch); //if the character is lower case,convert it to upper case.

}

if(isdigit(ch)|| (ch == '\_'))

{

control\_var = 1; //this variable , when 1 means the word is an identifier.

}

words[a] = ch;

ch = getc(filePsi);

a++;

}

words[a] = '\0';

identifier\_length = a;

if(control\_var == 0) //if the identifier is all letters, compared to keywords.

{

while(b< 18)

{

if(strcmp(words, keywords[b])== 0)

{

fputs("Keyword (", fileLex);

control\_var = 0;

break;

}

else

{

control\_var = 1;

}

b++;

}

}

if(control\_var == 1)

{

fputs("Identifier (", fileLex);

}

b = 0; //b is zeroes so as to be used again.

while(b<identifier\_length)

{

fputc(words[b], fileLex); //the identifier character by character is put in the fileLex.

b++;

}

fputs(")\n", fileLex);

}

//operators control.

if(ch == '+') // checking for the + operator.

{

ch = getc(filePsi);

if(ch =='+') // if after the first + operator comes another +.

{

fputs("Operator (++)\n", fileLex); // Operator ++ found.

}

else

{

fputs("Operator(+)\n", fileLex); //else operator + found.

continue;

}

}

if(ch == '-') // checking for the - operator.

{

ch = getc(filePsi);

if(ch == '-') // if after the first + operator comes another -.

{

fputs("Operator(--)\n", fileLex); // Operator -- found.

}

else

{

fputs("Operator(-)\n", fileLex); //else operator - found.

continue;

}

}

if(ch == ':') // checking for := operator.

{

ch = getc(filePsi);

if(ch == '=')

{

fputs("Operator(:=)\n", fileLex);

}

else

{

fputs("Undefined input :\n", fileLex);

continue;

}

}

if(ch == '\*')

{

fputs("Operator(\*)\n", fileLex);

}

if(ch == '/')

{

fputs("Operator(/)\n", fileLex);

}

//comments ,LeftPar and RightPar control.

if(ch == '(')

{

ch = getc(filePsi);

if(ch == '\*')

{

while(ch = getc(filePsi)!= EOF)

{

ch = getc(filePsi);

if(ch == '\*')

{

ch = getc(filePsi);

if(ch == ')')

{

ch = getc(filePsi);

break;

}

if(ch == EOF)

{

fputs("Lexical Error.The comment didn't end before EOF.\n", fileLex);

}

}

}

}

else

{

fputs("LeftPar\n ", fileLex);

continue;

}

continue;

}

if(ch == ')')

{

fputs("RightPar\n", fileLex);

}

//strings control.

if(ch == '"')

{

ch = getc(filePsi);

strings[k] = ch; // every character read after the " is added to the strings array.

k++;

while(ch != '"' && ch != EOF)

{

ch = getc(filePsi);

if(ch == '"') // when the character is equal to " the string is terminated.

{

break;

}

strings[k] = ch;

k++;

}

if(ch == '"') //if second " is reached than the string is terminated.

{

fputs("String (", fileLex);

while(m<strlen(strings))

{

fputc(strings[m], fileLex);

m++;

}

fputs(")\n", fileLex);

}

else if (ch == EOF)

{

fputs("Lexical Error occurred. String didn't finish before EOF.", fileLex);

}

}

//integer control.

if(isdigit(ch))

{

fputs("IntConst(", fileLex);

while (isdigit(ch))

{

if(int\_len > 10)

{

fputs("Integers can't be longer than 10 characters!", fileLex);

return 1;

}

fputc(ch , fileLex);

ch = getc(filePsi);

int\_len++; //counts the total number of digits.

}

fputs(")\n", fileLex);

int\_len = 0;

continue;

}

//brackets and end of line control.

if(ch == ';')

{

fputs("End Of Line\n", fileLex);

}

if(ch == '[')

{

fputs("LeftSquareBracket.\n", fileLex);

}

if(ch == ']')

{

fputs("RightSquareBracket.\n", fileLex);

}

if(ch == '{')

{

fputs("LeftCurlyBracket.\n", fileLex);

}

if(ch == '}')

{

fputs("RightCurlyBracket.", fileLex);

}

ch = fgetc(filePsi);

}

}

fclose(filePsi); //filePsi closed.

fclose(fileLex); //fileLex closed.

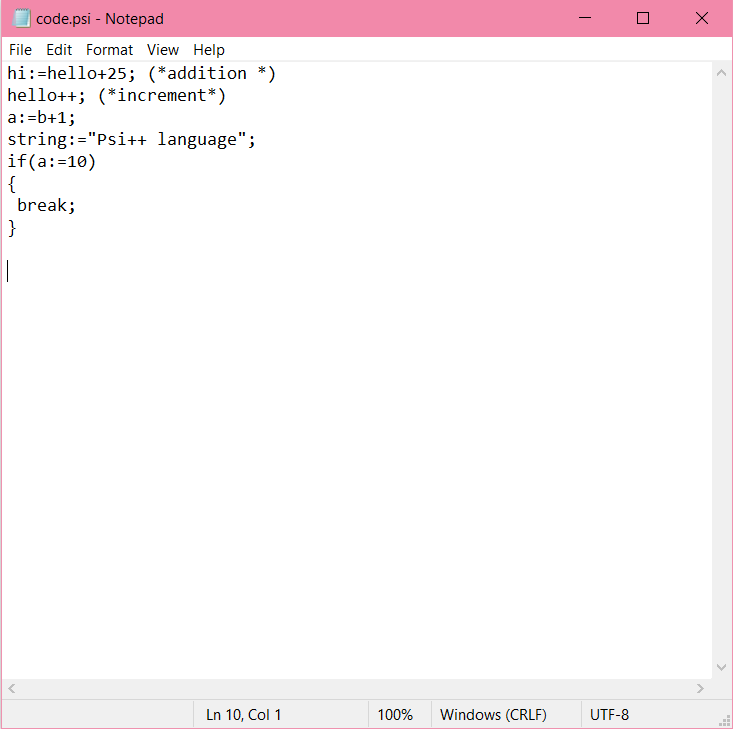
return 0;

}

**The analysis**

* In this program first the code.psi file and code.lex files were opened.
* After that if the file we want was found , begins the analyze of the characters.
* First we checked if the character was a space. If it was ,would not be considered.
* Secondly to be analyzed were the identifiers. If the character read from the file was a letter , lower or upper case, and if it included digits or \_ (underscore) the characters were added to the WORDS array. After that the word was compared with each of the elements of the array KEYWORDS to check if the word was a keyword. After that it was printed in the code.lex file.
* Thirdly the operators were analyzed. Using if’s and if true written to the code.lex file with the proper way.
* The analyze continues with the comments. Here included were the left and right parantheses because of being related with the comments syntax.
* For the strings if the character read was “ , then begins its analysis. The moment that another “ character was read the string finishes and is written to the proper file.
* Next , integer constants were analyzed. With how many digits there were it is written to the code.lex file.
* And at last the brackets control is done by using if’s and writing the tokens in the code.lex file.

The code.psi file



The output of the program.

