CSE – 430

Compiler Design Lab

Project Report

Topic: Creating Lexical analyzer using C Language.

Prepared By:

Mohammad Enan Al Harun Sahan

Reg No. 20101095

Section: B2

MD. Asadujjaman Noor

Reg No. 20101101

Section: B2

Sheikh Nafez Sadnan

Reg:20101106

Section: B2

Presented to:

Fabliha Haque

Lecturer, CSE, UAP

Project Title:

Creating a Lexical Analyzer using C language.

Code Explanation:

Input File:

First, we initiate our main function and open our input file. Here the input is “demo.txt” file.

PseudoCode:

```
Main function:
    Declare variables ch, temp[40], arithmetic_operator, pun.
    Declare file_pointer.
    Declare count and initialize x to 0 as integers.

    Open a file named "demo.txt" in read mode and assign it to file_pointer.
    If file_pointer is NULL, print "file not found." and exit.
```

Ignore Comments:

Then we implement the logic of processing characters from the file, checking for comments to ignore.

PseudoCode:

```
Set comment_flag to 0.
While there are still characters to read from file_pointer:
    Read the next character from file_pointer into ch.
    If comment_flag is 1 and ch is not a newline character:
```

```
    Continue to the next iteration of the loop.
Else:
    Set comment_flag to 0.


If ch is a forward slash '/':
    Read the next character from file_pointer into ch.
    If ch is also a forward slash '/':
        Set comment_flag to 1.
    Else:
        Add the forward slash '/' to the set arith.
```

Arithmetic Operators:

Then next part of our code detects arithmetic operators, and increments the counter after executing the loop.

PseudoCode:

```
Set count to 0.
While count is less than 6:
    If ch is equal to the character at index count in arithmetic_operator:
        Add ch to the set arith.
        Add the contents of temp to the set iden_string.
        Set temp[x] to null character '\0'.
        Set x to 0.
        Set token_p to the contents of temp.
        Break from the loop.
    Increment count by 1.
```

Punctuation:

In this part we outline the logic of processing characters from the file, checking for punctuation characters, and adding alphanumeric characters to the temp array.

PseudoCode:

```
Set count to 0.
While count is less than 3:
    If ch is equal to the character at index count in pun:
        Add ch to the set pun_string.
    Increment count by 1.
If ch is alphanumeric:
    Add ch to the temp array at index x.
    Increment x by 1.
```

Functions and Parenthisis:

We implemented the logic of checking for specific characters and handling them accordingly, including checking if a token precedes a parenthesis and categorizing it as a function if certain conditions are met.

PseudoCode:

```
If ch is any of '(', ')', '{', '}', '[', or ']':
    If ch is '(' and token_p is "int", "void", "float", or "double":
        Set the character at index x in the temp array to null character '\0'.
        Add the contents of temp to the set fun.
    Reset x to 0.
    Add ch to the set parenthesis.
```

Keywords:

We made a keyword_library function, which checks if a given string is a keyword by comparing it with a predefined list of keywords.

PseudoCode:

```
Function keyword_library(temp):
  Set count to 0 and flag to 0.
  Define a 2D array keywords with 32 rows and 12 columns, initialized with keywords.
  Loop while count is less than or equal to 31:
    If the string at index count in keywords is equal to temp:
      Set flag to 1.
      Break from the loop.
    Increment count by 1.
  Return flag.
```

Constants:

This pseudocode outlines the logic of the const_library function, which checks if a given string consists solely of digits. If so, it returns 1; otherwise, it returns 0.

PseudoCode:

```
Function const_library(temp):
  Set count to 0, i to 0, and flag to 0.
  Define an array constant containing the digits '0' to '9'.
  Initialize size to 0.
  While temp[size] is not null character '\0':
    Increment size.
  While temp[i] is not null character '\0':
    Set count to 0.
```

```
    Loop while count is less than 10:
        If constant[count] is equal to temp[i]:
            Increment flag by 1.
            Break from the loop.
        Increment count by 1.
    Increment i by 1.
  If flag is equal to size:
    Return 1.
  Return 0.
```

Categorizing keywords/constants/header strings/identifiers:

We implemented the logic of processing characters from the file, checking for whitespace, and categorizing tokens into keywords, constants, header strings, or identifiers.

PseudoCode:

```
If ch is a newline or space character and x is not 0:
  Set the character at index x in the temp array to null character '\0'.
  Reset x to 0.
  Set token_p to the contents of temp.
  If keyword_library(temp) returns 1:
    Add temp to the keyword set.
  Else if const_library(temp) returns 1:
    Add temp to the constant set.
  Else:
    Calculate the length of temp and store it in len.
    Set ok to 0.
    Set in to the string "include".
    Loop from i = 0 to 6:
```

```
        If the character at index i in temp is equal to the character at index i in in, increment ok.
    If ok is equal to 7 and the last character of temp is 'h':
        Initialize a string ans to an empty string.
        Loop from i = 7 to len - 2:
            Append the character at index i of temp to ans.
        Append ".h" to ans.
        Add ans to the header_string set.
    Else:
        Add temp to the iden_string set.
```

Output:

This part outlines outputting the contents of various sets along with their sizes and then closing the file pointer.

PseudoCode:

```
Output "Identifier (" followed by the size of iden_string, then ": ".
For each item i in iden_string:
    Output i followed by a space.
Output a newline.

Output "Punctuation (" followed by the size of pun_string, then ": ".
For each item i in pun_string:
    Output i followed by a space.
Output a newline.

Output "Constant (" followed by the size of constant, then ": ".
For each item i in constant:
    Output i followed by a space.
```

```
Output a newline.

Output "Parenthesis (" followed by the size of parenthesis, then ": ".
For each item i in parenthesis:
    Output i followed by a space.
Output a newline.

Output "Arithmetic (" followed by the size of arith, then ": ".
For each item i in arith:
    Output i followed by a space.
Output a newline.

Output "Keyword (" followed by the size of keyword, then ": ".
For each item i in keyword:
    Output i followed by a space.
Output a newline.

Output "Header (" followed by the size of header_string, then ": ".
For each item i in header_string:
    Output i followed by a space.
Output a newline.

Output "Functions (" followed by the size of fun, then ": ".
For each item i in fun:
    Output i followed by a space.
Output a newline.

Close the file_pointer.
Return 0.
```

Conclusion:

Code Snippet:

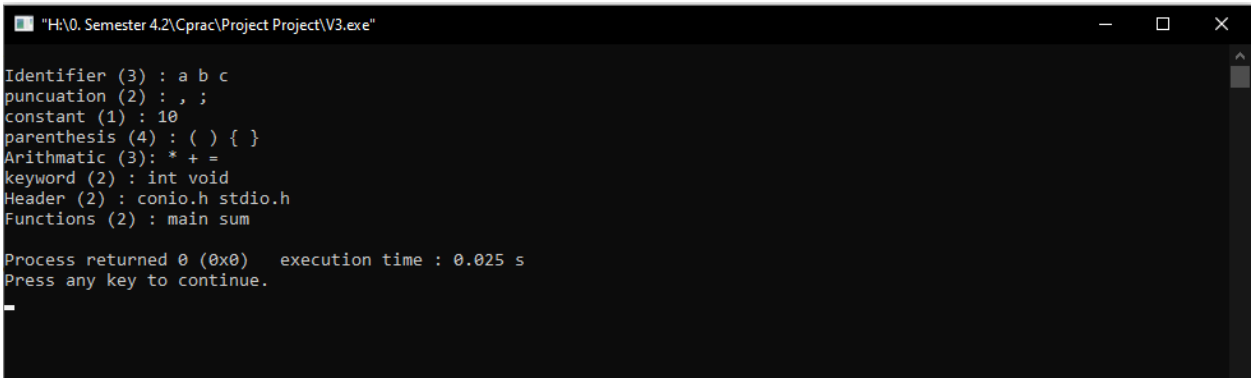
V3.cpp - Code::Blocks 20.03

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

Start here | V3.cpp

```cpp
#include<bits/stdc++.h>
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<string.h>
#include<stdlib.h>
using namespace std;



stack<char>st;
set<string> constant , keyword,header_string,iden_string,fun;
set<char>arith, pun_string,parenthesis;
string token_p;

int keyword_library(char temp[]);
int const_library(char tem[]);
int main()
{
	char ch, temp[40], arithmetic_operator[] = "=+%*/-" , pun[]=";:,";
	FILE *file_pointer;
	int count, x = 0;
	file_pointer = fopen("demo.txt","r");
	if(file_pointer == NULL)
	{
		printf("file not found.\n");
		exit(0);
	}
	int comment_flag=0;
	while((ch = fgetc(file_pointer)) != EOF)
	{
		if (comment_flag==1 && ch!='\n')
		continue;
		else comment_flag=0;

		if(ch=='/')
		{
		  ch = fgetc(file_pointer);
		  if(ch=='/')
		    comment_flag=1;
		  else
		  arith.insert('/');
		}

		count = 0;
		while(count < 6)
		{
			if(ch == arithmetic_operator[count])
			{
				arith.insert(ch);
				iden_string.insert(temp);
				temp[x] = '\0';
				x=0;
				token_p=temp;
				break;
			}
```

Output Snippet:



Our lexical analyzer is able to determine **“Identifiers”, “Punctuations”, “Constants”, “Parenthesis”, “Arithmetic”, “Keywords”, “Headers” and “Functions”**. It also accounts the number of times these items appear individually. Thus, we were able to meet the project conditions and complete our project.