Assignment No 1

CSE-0302 Summer 2021

Afrin Sultana
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
afrinsultana.su@gmail.com

***Abstract*—Lexical Analysis. This works denotes filtering comments and white space characters from a source and another one works as reads any simple program as source and separates out the valid tokens from the source program.**

n

***Index Terms*—The word mostly used in my report. C,C++,Lexical Analysis.**

I. Introduction

In this program we need to remove the comments and white space (extra spaces, tabs and newline characters).And from this we can understand How can a computer remove the comments and white space we can understand about how to work compiler also from this program. Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks syntaxes into a series of tokens, by removing any whitespace or comments in the source code. The name ”compiler” is primarily used for programs that translate source code from a high-level programming language to a lower level language. The main task of lexical analysis is to read input characters in the code and produce tokens

II. Literature Review

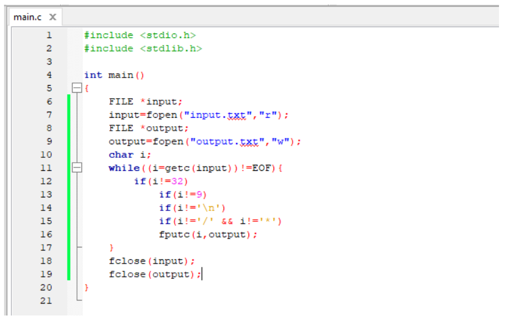
Context-Sensitive Analysis Keith D. Cooper, Linda Torczon, in Engineering a Compiler (Second Edition), 2012 In this, examined two ways to perform context-sensitive analysis: attribute-grammar formalism and an ad hoc approach. For context-sensitive analysis, unlike scanning and parsing, formalism has not displaced the ad hoc approach. Regular Expressions and Languages Raymond Greenlaw, H. James Hoover, in Fundamentals of the Theory of Computation: Principles and Practice, 1998 It all attention to the lex command.The purpose of lex is to generate lexical analyzers.

III. Proposed Methodology

The methodology you work, explain here with code and other items.

IV. Conclusion and Future Work

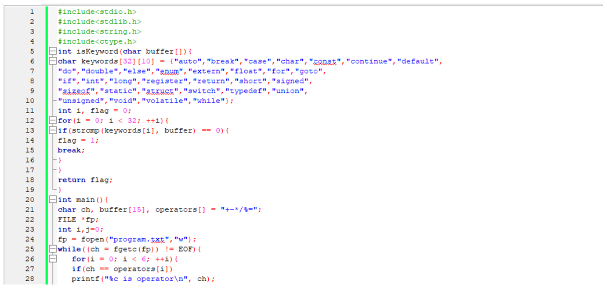
In future, we use frame work and use this for making another big projects.



main.c

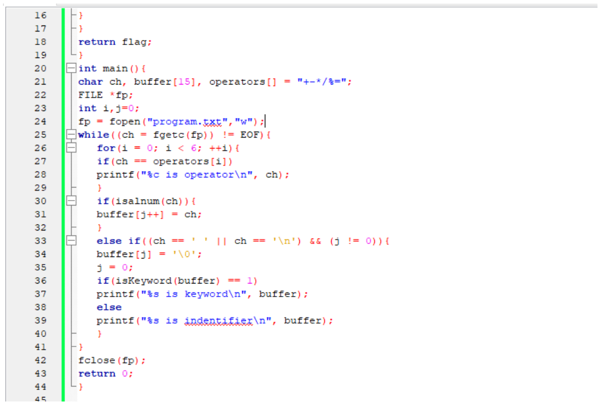
```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    FILE *input;
    input=fopen("input.txt","r");
    FILE *output;
    output=fopen("output.txt","w");
    char i;
    while((i=getc(input))!=EOF){
        if(i!=32)
            if(i!=9)
            if(i!='\n')
            if(i!='/' && i!='*')
            fputc(i,output);
    }
    fclose(input);
    fclose(output);
}
```

Fig. 1. Example of a Scanning and Filtering a Source Program .



```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
int isKeyword(char buffer[]){
char keywords[32][10] = {"auto","break","case","char","const","continue","default",
"do","double","else","enum","extern","float","for","goto",
"if","int","long","register","return","short","signed",
"sizeof","static","struct","switch","typedef","union",
"unsigned","void","volatile","while"};
int i, flag = 0;
for(i = 0; i < 32; ++i){
if(strcmp(keywords[i], buffer) == 0){
flag = 1;
break;
}
}
return flag;
}
int main(){
char ch, buffer[15], operators[] = "+-*/%=";
FILE *fp;
int i,j=0;
fp = fopen("program.txt","w");
while((ch = fgetc(fp)) != EOF){
   for(i = 0; i < 6; ++i){
   if(ch == operators[i])
   printf("%c is operator\n", ch);
```

Fig. 2. Example of a Lexical Analysis 1 .



```c
}
}
return flag;
}
int main(){
char ch, buffer[15], operators[] = "+-*/%=";
FILE *fp;
int i,j=0;
fp = fopen("program.txt","w");
while((ch = fgetc(fp)) != EOF){
   for(i = 0; i < 6; ++i){
   if(ch == operators[i])
   printf("%c is operator\n", ch);
   }
   if(isalnum(ch)){
   buffer[j++] = ch;
   }
   else if((ch == ' ' || ch == '\n') && (j != 0)){
   buffer[j] = '\0';
   j = 0;
   if(isKeyword(buffer) == 1)
   printf("%s is keyword\n", buffer);
   else
   printf("%s is indentifier\n", buffer);
   }
}
fclose(fp);
return 0;
}
```

Fig. 3. Example of a Lexical Analysis 2 .

Acknowledgment

I would like to thank my honourable**Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

References

[1] Hanks, P. (2013). Lexical analysis: Norms and exploitations. Mit Press.
[2] Wilhelm, R., & Maurer, D. (1995). Compiler design. Reading: Addison-Wesley Publishing Company.
[3] Beverly, R., & Bauer, S. (2005, July). The Spoofer project: Inferring the extent of source address filtering on the Internet. In Usenix Sruti (Vol. 5, pp. 53-59).
[4]
[5]
[6]
[7]