



Programming

Languages Translation

Flex : Fast lexical analyzer generator

| Name | ID |
|----------------|------|
| Mohamed Harraz | 4608 |
| Sherif Rafik | 4635 |
| Omar Nasr | 4730 |

Introduction:

Flex (fast lexical analyzer generator) is a tool/program for generating lexical analyzers (scanners or lexers).

Flex was written in C around 1987 by Van Paxson, with the help of many ideas and inspirations from Van Jacobson.

It is usually used together with **Berkeley Yacc parser generator** or **GNU Bison parser generator**.

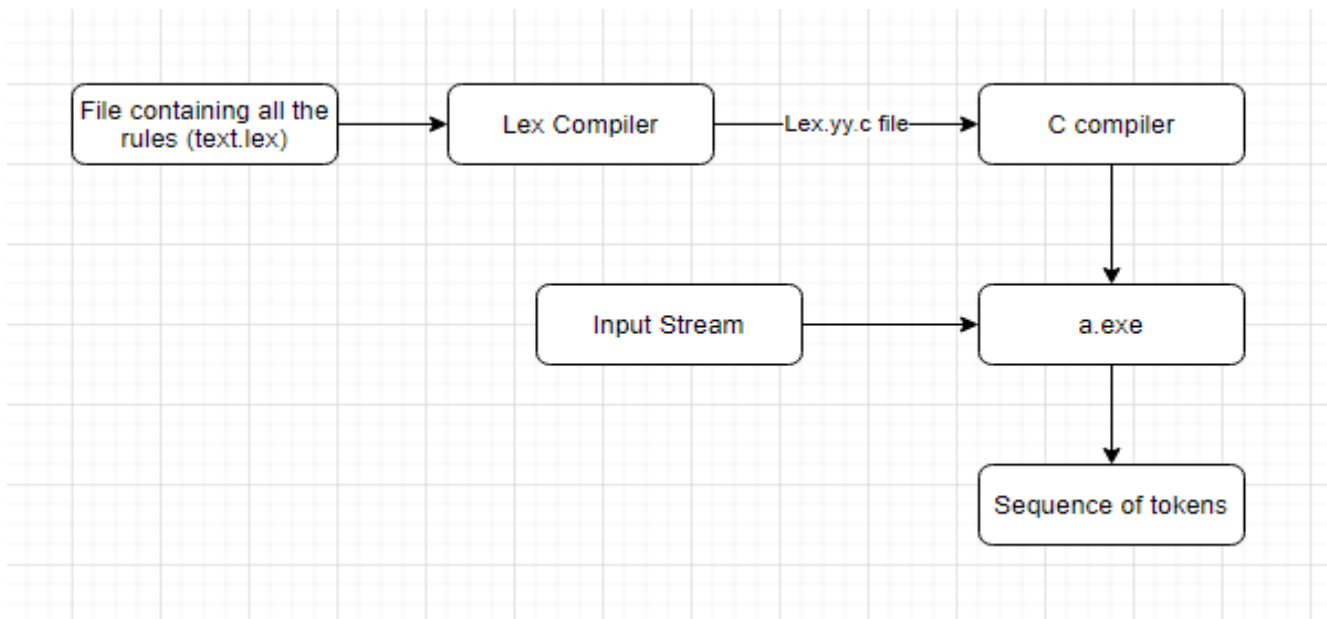
Flex can only generate code for **C** and **C++**.

Supported languages:

Flex can only generate code for **C** and **C++**.

To use the scanner code generated by flex from other languages a language binding tool such as **SWIG** can be used.

How does flex work?



The function **yylex()** is the main flex function which runs the rules section, and the extension **(.l or .lex)** is the extension used to save the programs.

- **Step 1:**

An input file describes the lexical analyzer to be generated written in a file with **.l or .lex** extension, and written in the lex language. The lex compiler transforms this file to a C program, in a file always named **lex.yy.c**.

- **Step 2:**

The C compiler compiles **lex.yy.c** file into an executable file a.exe.

- **Step 3:**

The output file a.exe takes an input stream and produces a stream of tokens.

Program structure:

In the input file, there are 3 sections

1. Definition Section: This section is optional, it contains the declarations of variables, regular definitions, manifest constants. In this section, the text is enclosed by %{ %} brackets.

- **Syntax:**

```
%{  
  
    Definitions  
  
%}
```

2. Rules Section: This section is required, it contains a series of rules in the form of patterns and actions. Actions must be on the same line of the corresponding patterns. In this section, the text is enclosed by %% %%

- **Syntax:**

```
%%  
  
Pattern Action  
  
%%
```

3. User Code Section: This section is optional, it contains C statements and additional functions. Two functions are called in this section **yywrap()** and **yylex()**

Example using given rules:

```
%{
    // Definitions section
    // Optional
}%

digit      [0-9]
letter     [a-zA-Z]
digits     {digit}+

%%

    // Rules section
    // Pattern      Action
"boolean"   {printf("boolean\n");}
"int"       {printf("int\n");}
"float"     {printf("float\n");}
"if"        {printf("if\n");}
"else"      {printf("else\n");}
"while"     {printf("while\n");}
"+"         {printf("addop\n");}
"-"         {printf("addop\n");}
"*"         {printf("mulop\n");}
"/"         {printf("mulop\n");}
"="         {printf("assign\n");}
"=="        {printf("relop\n");}
"!="        {printf("relop\n");}
">"         {printf("relop\n");}
">="        {printf("relop\n");}
"<"         {printf("relop\n");}
"<="        {printf("relop\n");}
";"         {printf(";\n");}
","         {printf(",\n");}
"("         {printf("(\n");}
")"         {printf(")\n");}
"}"         {printf("}\n");}
"{"         {printf("{\n");}
{letter}{letter}|{digit}* {printf("id\n");}
({digit}+)|({digit}+"."{digits}("E"{digits})?) {printf("num\n");}

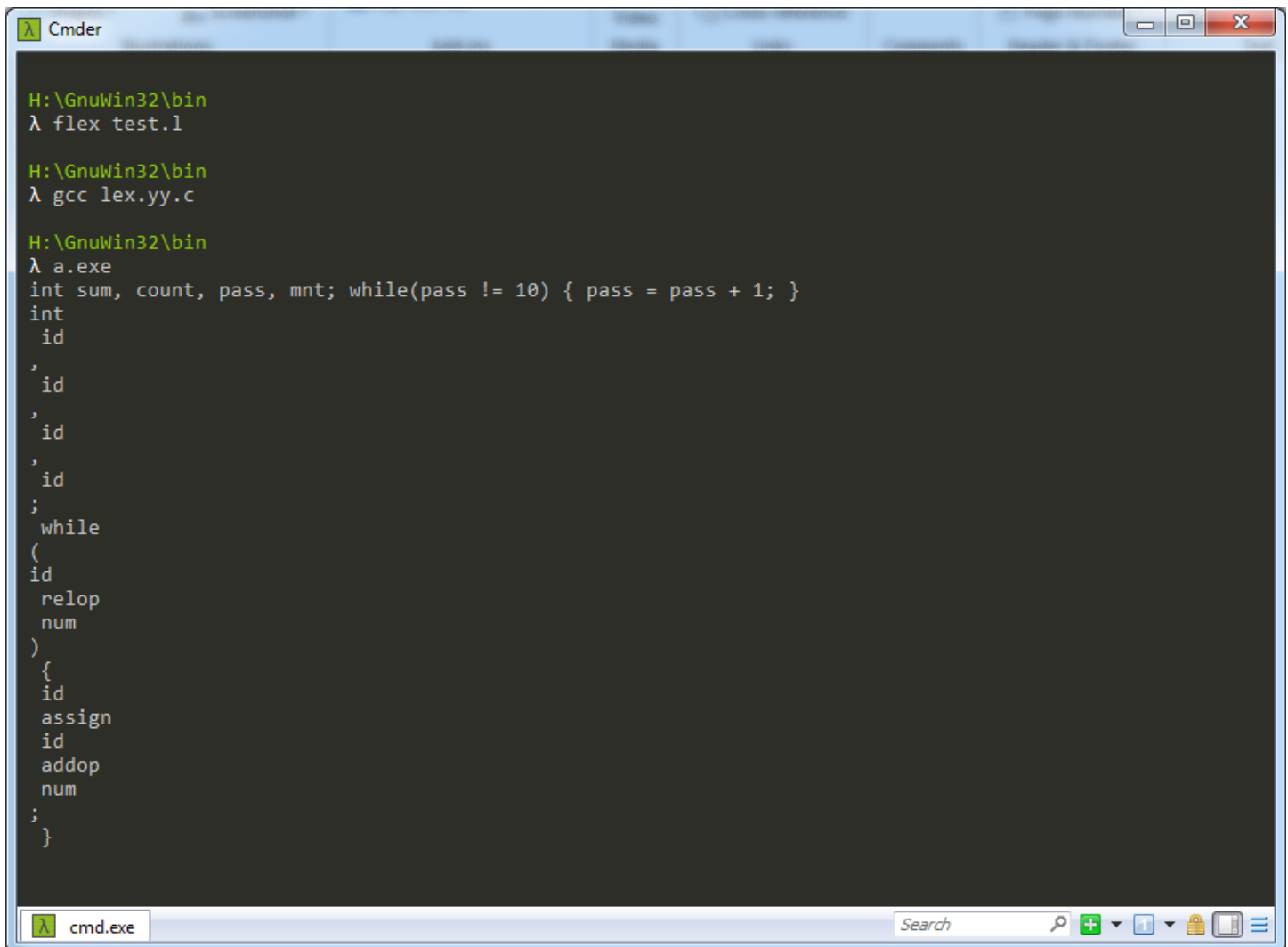
%%

// USER CODE SECTION
int yywrap(){}
int main(){
    yylex();
    return 0;
}
```

Program:

```
int sum, count, pass,  
mnt; while(pass != 10)  
{  
    pass = pass + 1;  
}
```

Output:



The screenshot shows a Windows Command Prompt window titled 'Cmder'. The user is in the directory 'H:\GnuWin32\bin'. They enter the command 'flex test.l', followed by 'gcc lex.yy.c', and then 'a.exe'. The output of the program is displayed, showing the initial state of variables and the execution of the while loop.

```
H:\GnuWin32\bin  
λ flex test.l  
  
H:\GnuWin32\bin  
λ gcc lex.yy.c  
  
H:\GnuWin32\bin  
λ a.exe  
int sum, count, pass, mnt; while(pass != 10) { pass = pass + 1; }  
int  
id  
,  
id  
,  
id  
,  
id  
;  
while  
(  
id  
relop  
num  
)  
{  
id  
assign  
id  
addop  
num  
;  
}
```