Fourth Year Laboratory

E4 - Building a compiler using Flex and Bison

Amr Keleg

Faculty of Engineering, Ain Shams University

March 30, 2021

Contact: amr_mohamed@live.com

Outline

1 Introduction
- Installation steps
- Basic idea

2 Flex

3 Bison

Outline

1 Introduction
- Installation steps
- Basic idea

2 Flex

3 Bison

For Windows

- Use the exe files here: `https://drive.google.com/file/d/1w6IX5vwkeTGn15YAvcR53dMDZo4bKUpR`
- You will need a C IDE (Visual Studio or CodeBlocks or any other IDE/C compiler)

For Ubuntu

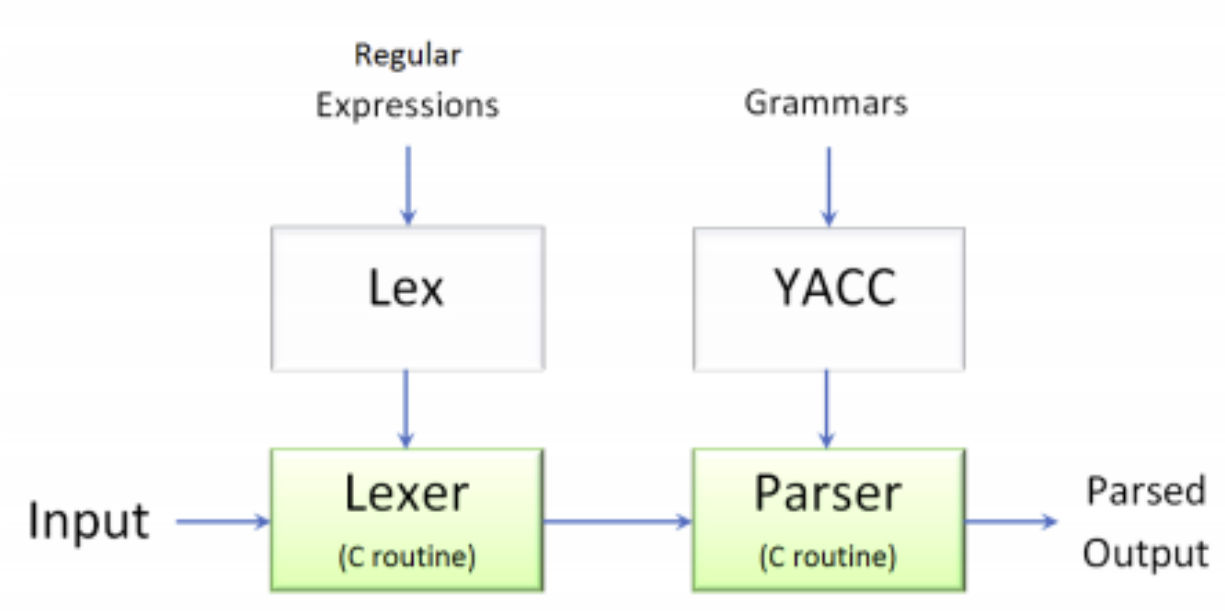
- `$ sudo apt install gcc flex bison`
- (Optional) C IDE (e.g: Codeblocks)
- Ensure the programs are properly installed using:
  - `$ gcc -v`
  - `$ flex -V`
  - `$ bison -V`

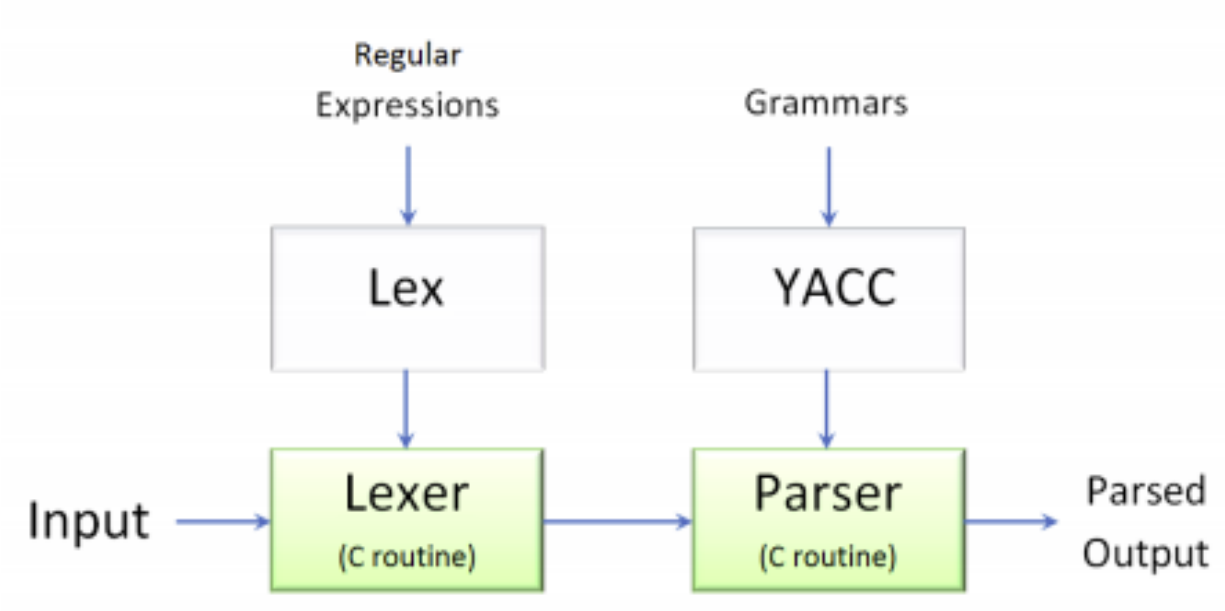
Outline

1 Introduction
- Installation steps
- Basic idea

2 Flex

3 Bison


Regular
Expressions
Grammars
Lex
YACC
Input
Lexer
(C routine)
Parser
(C routine)
Parsed
Output



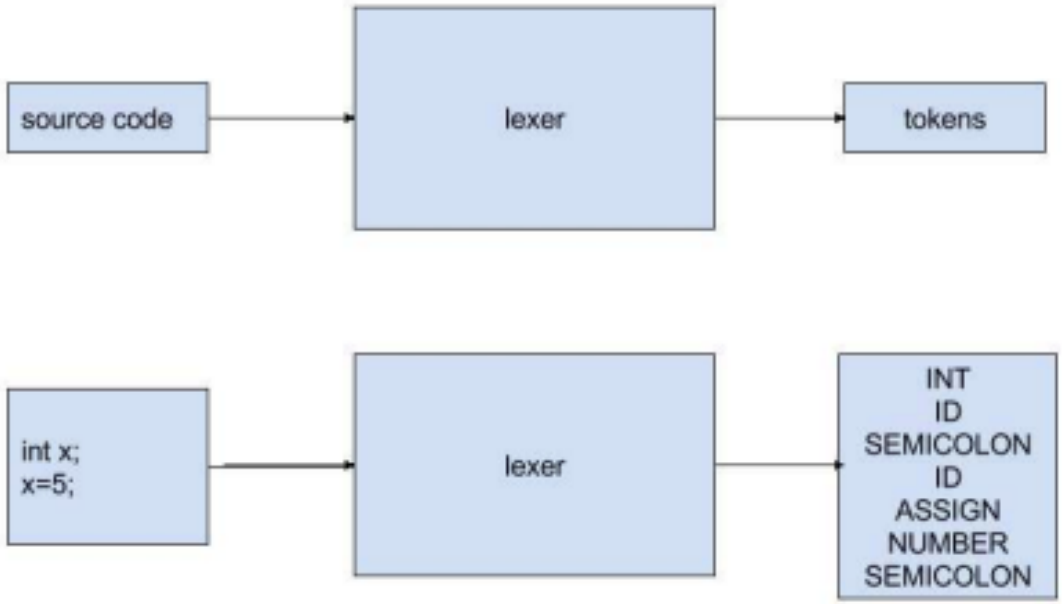
The GNU open-source alternatives of "Lex and Yacc" are "Flex and Bison"

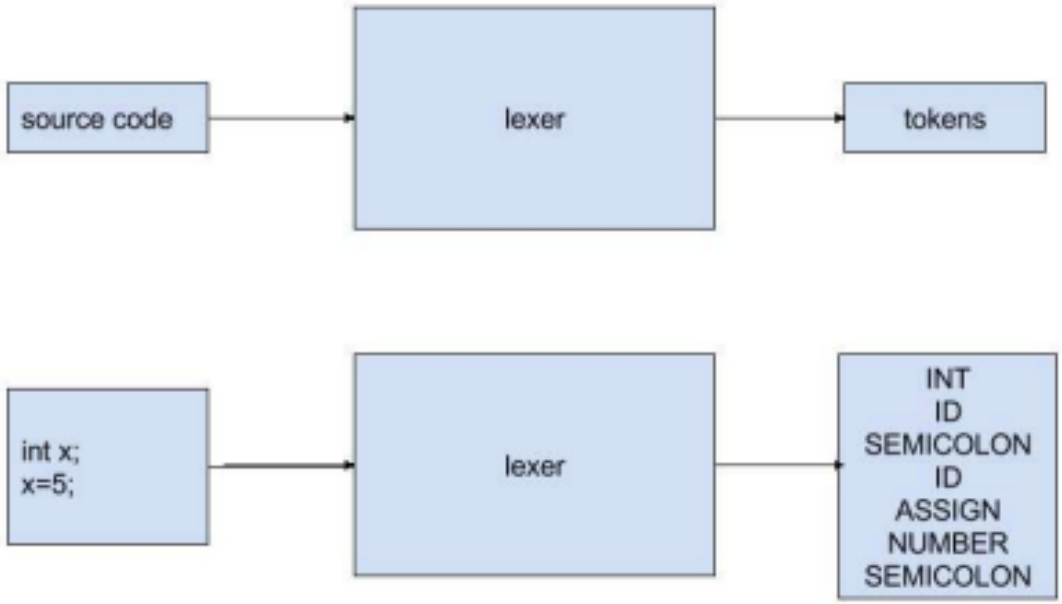
Outline

1 Introduction

2 Flex

3 Bison


source code
lexer
tokens
int x;
x=5;
lexer
INT
ID
SEMICOLON
ID
ASSIGN
NUMBER
SEMICOLON



Flex makes use of **Regular Expressions** 😍 to build lexers (scanners)

- `digit 0|1|2|3|4|5|6|7|8|9`

---

https://amr-keleg.github.io/blog/2020/regex-cheatsheet/

- `digit 0|1|2|3|4|5|6|7|8|9`
- `digit [0123456789]`
  - This is called a character group
  - Why does it differ from the first regex?

---

https://amr-keleg.github.io/blog/2020/regex-cheatsheet/

- `digit 0|1|2|3|4|5|6|7|8|9`
- `digit [0123456789]`
  - This is called a character group
  - Why does it differ from the first regex?
- `digit [0-9]`
  - Character groups allow ranges using the - symbol
  - Any special token in a character group doesn't need escaping
  - e.g: [.*] matches a single dot or a single asterisk

---

https://amr-keleg.github.io/blog/2020/regex-cheatsheet/

- `digit 0|1|2|3|4|5|6|7|8|9`
- `digit [0123456789]`
  - This is called a character group
  - Why does it differ from the first regex?
- `digit [0-9]`
  - Character groups allow ranges using the - symbol
  - Any special token in a character group doesn't need escaping
  - e.g: [.*] matches a single dot or a single asterisk
- Regex for identifier:

---

https://amr-keleg.github.io/blog/2020/regex-cheatsheet/

- `digit 0|1|2|3|4|5|6|7|8|9`
- `digit [0123456789]`
  - This is called a character group
  - Why does it differ from the first regex?
- `digit [0-9]`
  - Character groups allow ranges using the - symbol
  - Any special token in a character group doesn't need escaping
  - e.g: [.*] matches a single dot or a single asterisk
- Regex for identifier: `id [a-zA-Z_][a-zA-Z_0-9]*`

---

https://amr-keleg.github.io/blog/2020/regex-cheatsheet/

```
%{
#define YYSTYPE char*
#include "stdlib.h"
int lineno=1;
%}
%option nounistd
%option noyywrap
%option never-interactive
white [ \r\t]+
letter [A-Za-z]
digit [0-9]
id {letter}({letter}|{digit})*
number {digit}+
%%
{white} { }
{number} {printf("Number\n");}
"int" {printf("INT\n");}
{id} { printf("ID\n");}
"=" {printf("ASSIGN\n");}
%%
int yyerror(void)
{
    printf("Error\n");
}
int main()
{
    FILE* fp=fopen("D:\\lab4folders\\compiler\\test.txt","r");
    yyin=fp;
    yylex();
    system("pause");
}
```

```
%{
#define YYSTYPE char*
#include "stdlib.h"
int lineno=1;
%}
%option nounistd
%option noyywrap
%option never-interactive
white [ \r\t]+
letter [A-Za-z]
digit [0-9]
id {letter}({letter}|{digit})*
number {digit}+
%%
{white} { }
{number} {printf("Number\n");}
"int" {printf("INT\n");}
{id} { printf("ID\n");}
"=" {printf("ASSIGN\n");}
%%
int yyerror(void)
{
    printf("Error\n");
}
int main()
{
    FILE* fp=fopen("D:\\lab4folders\\compiler\\test.txt","r");
    yyin=fp;
    yylex();
    system("pause");
}
```

Why isn't **int** parsed as an ID?

Outline

1 Introduction

2 Flex

3 Bison

- Based on CFG (Context-free Grammar).

- Based on CFG (Context-free Grammar).
- The basic usage is to ensure that the code is syntactically correct.

- Based on CFG (Context-free Grammar).
- The basic usage is to ensure that the code is syntactically correct.
- It can be used to generate syntax trees or to generate intermediate code.

```
%{
#define YYSTYPE char*
#include "exp2.tab.h"
#include "stdlib.h"

int lineno=1;
%}
%option nounistd
%option noyywrap
%option never-interactive

white [ \r\t]+
letter [A-Za-z]
digit [0-9]
id {letter}({letter}|{digit})*
number {digit}+
%%
{white} { }
{number} {yylval=strdup(yytext); return NUMBER;}
"int" {yylval=strdup(yytext); return INT;}
{id} { yylval=strdup(yytext); return ID;}
"=" {yylval=strdup(yytext);return ASSIGN;}

%%

   int yyerror(void)
   {
        printf("Error\n");

   }
```

```
%{#include <stdio.h>
#include <stdlib.h>
#define YYSTYPE char*
extern FILE* yyin;
%}
%token NUMBER
%token ID
%token INT
%token ASSIGN
%%
/*Reduction Rule for the whole Program*/
Program:  Declaration Statements {};

/*Reduction rules for Declarations at the beginning of the program*/
Declaration:|
Declaration INT ID  { printf("INT ID ");}
;

/*Reduction rule for the set of statements*/
Statements:|
Statements Statement {}
;

/*Reduction rule for arithmetic assignment statements*/
Statement:ID ASSIGN NUMBER{ printf("ID ASSIGN NUMBER");}
;
%%
int main()
{
        FILE* fp=fopen("D:\\lab4folders\\compiler\\test.txt","r");
        yyin=fp;
        if (yyparse()!=0) printf("Error found.\n");
        system("pause");
}
```

Live demos/tutorial about using Flex/Bison:

- `https://www.youtube.com/watch?v=54bo1qaHAfk`
- `https://www.youtube.com/watch?v=__-wUHG2rfM`