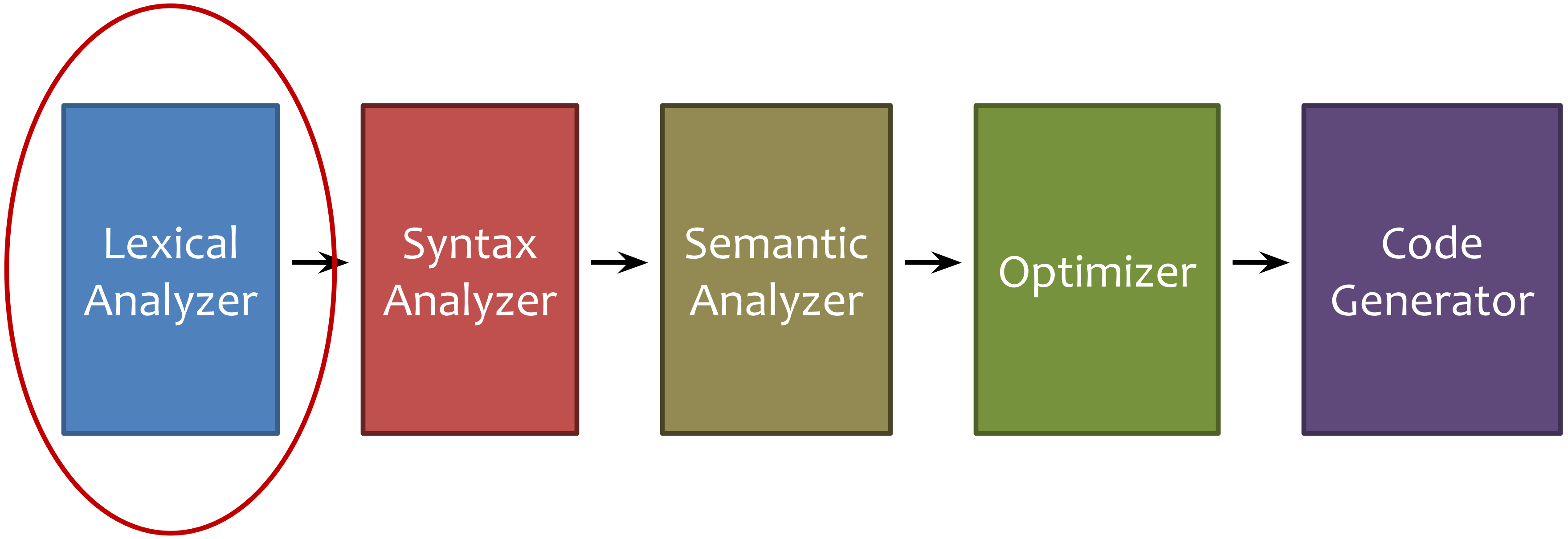
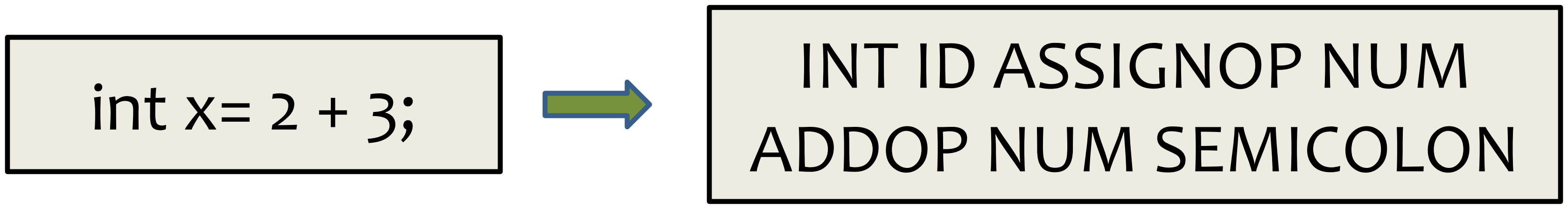
Lexical Analysis using FLEX

Compiler Overview



Lexical Analysis

- First phase of compilation
- Process of converting sequence of characters into sequence of tokens



```
int x= 2 + 3;
```

```
INT ID ASSIGNOP NUM
ADDOP NUM SEMICOLON
```

Role of Lexical Analyzer

- Identify Tokens
- Insert Lexemes into Symbol Table
- Remove all white spaces
- Return Tokens to Parser

Token and Lexeme

- A Token is a group of characters having collective meaning, typically a word or a punctuation mark

- A Lexeme is an actual character sequence forming a certain instance of a token

- For example, the number 23 in a source code is a lexeme and its corresponding token is INTEGER

How to build Lexer?

- From Scratch?
- No! There are tools that generate lexer.

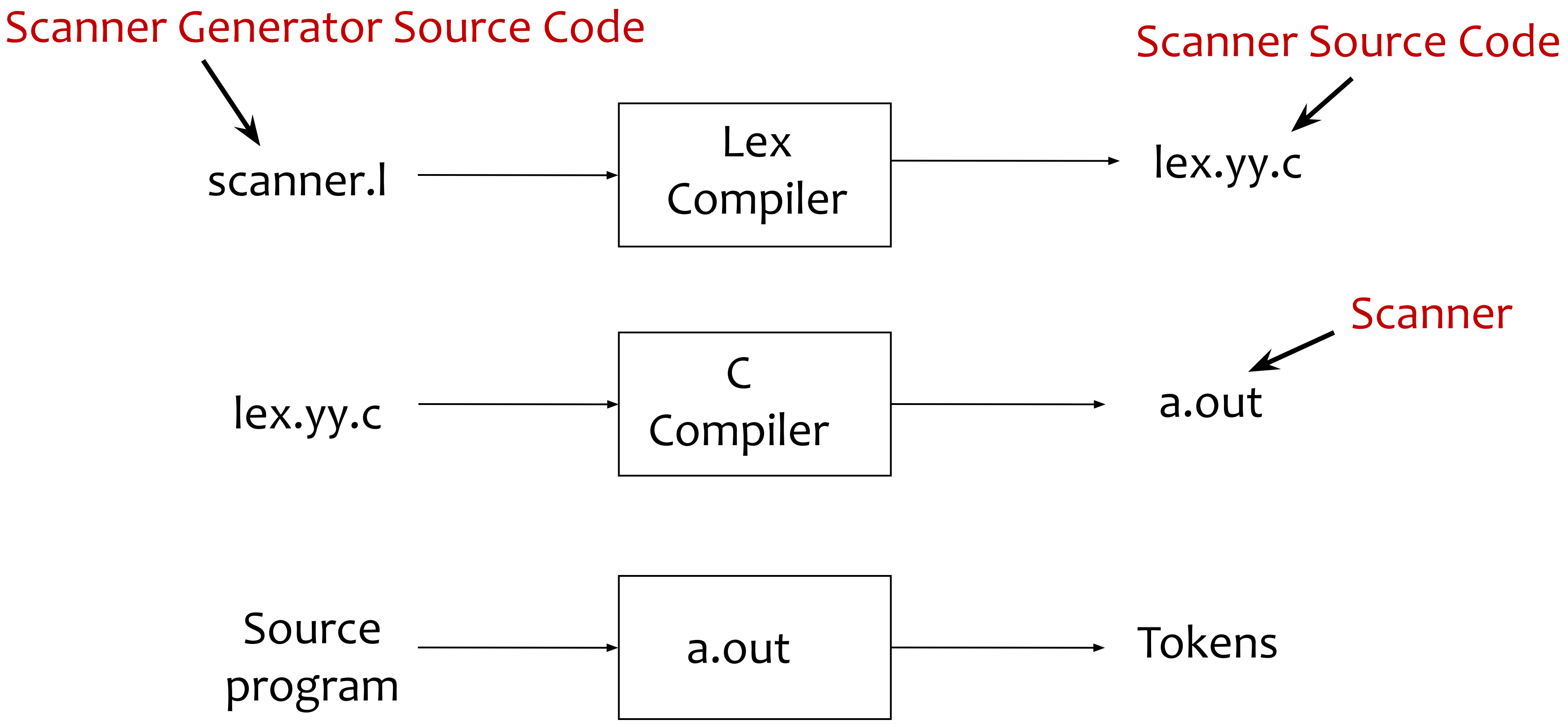
Meet the Life Savers

- lex
  - Lexical Analyzer Generator
  - Not used anymore
- flex
  - Free, open source alternative
  - We will use this

flex

- flex stands for Fast Lexical Analyzer Generator
- It is a tool for generating scanners
- flex reads the input file and tries to recognize lexical patterns in it
- You have to provide the rules to detect patterns

flex/lex


Scanner Generator Source Code
Scanner Source Code
scanner.l
Lex Compiler
lex.yy.c
Scanner
lex.yy.c
C Compiler
a.out
Source program
a.out
Tokens

flex Installation

- Run the following commands in the terminal

```
sudo apt-get update
sudo apt-get install flex
```

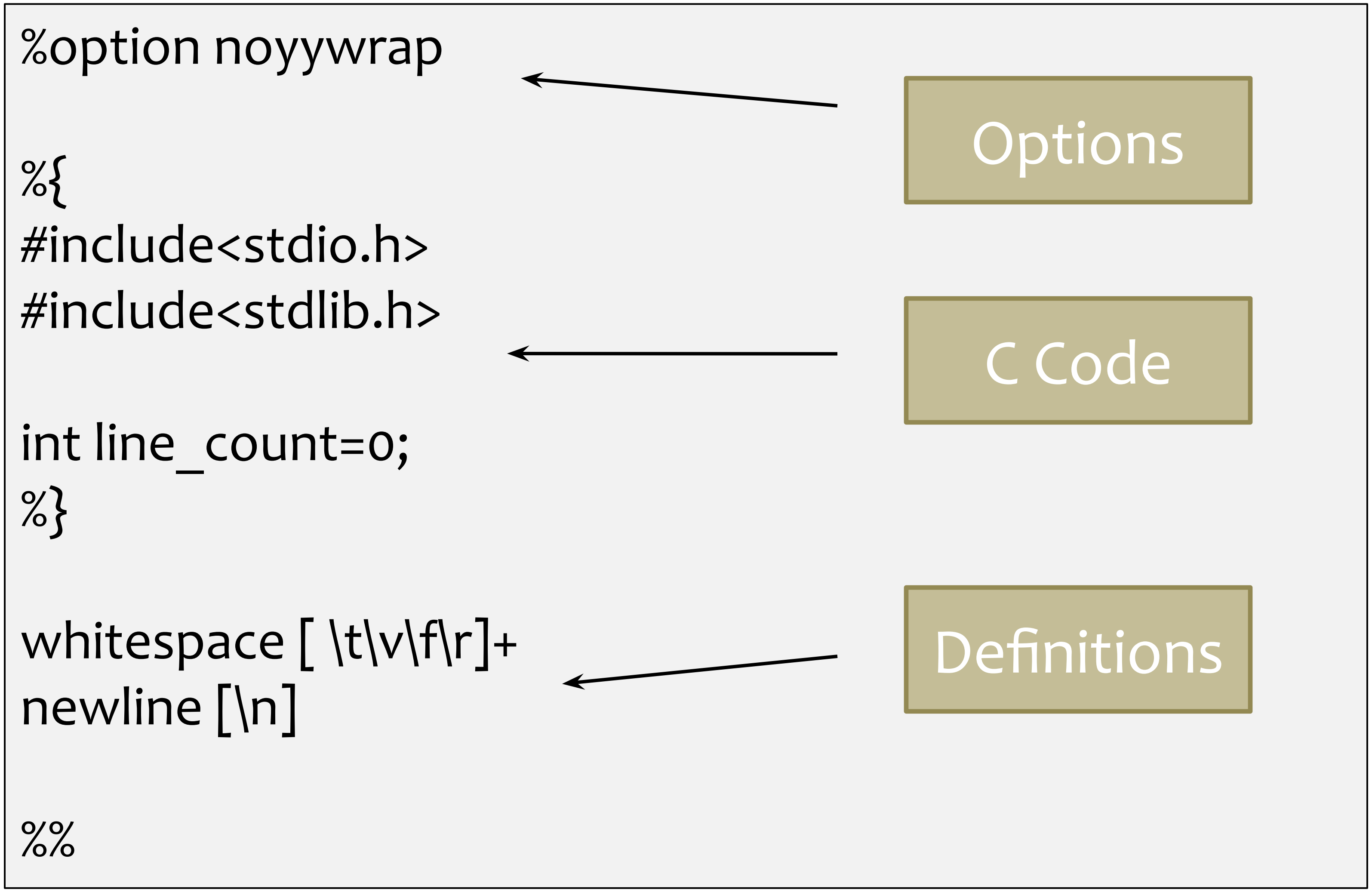
flex Program Structure

```
/**** Definition Section *****/

%%

/**** Rules Section ********/

%%

/**** User Subroutines ****/
```

i. Definition Section

- Definition Section typically includes
  - Options
  - C code to be copied in lex.yy.c
  - Definitions

i. Definition Section



```
%option noyywrap

%{
#include<stdio.h>
#include<stdlib.h>

int line_count=0;
%}

whitespace [ \t\v\f\r]+
newline [\n]


%%
```

ii. Rules Section

- Rules Section may include
  - Pattern Lines
  - C code to be copied in lex.yy.c
  -
- Usually it only contains some pattern lines with corresponding actions
- Remember that lexical patterns are matched starting from the topmost rule

ii. Rules Section

```
%%

[0-9]+        {printf("%s is a number",yytext);}
{whitespace}   {printf("whitespace encountered");}
{newline}      {line_count++;}
.           {printf("Mysterious character found");}

%%
```

Pattern

Action

Do not place any whitespace at the beginning of a pattern line

iii. Subroutine Section

- Subroutine section usually includes C code to be copied in lex.yy.c file

- If you want yywrap() or main(), then you should write it here

iii. Subroutine Section

```
%%

int main(int argc, char** argv) {
      yyin = fopen(argv[1], "r");
      yylex();
      fclose(yyin);
      return 0;
}
```

This function matches patterns

Regular Expressions

- Metacharacters (Characters with special meaning)

Metacharacter	Meaning	Example
[]	Match any character within this bracket	[abc] [a-z] [A-z] [-aZ]
{-} and {+}	Set Difference or Union	[a-z]{-}[aeiou]
*	Zero or more occurrence of preceding expression	a* 12*3
+	One or more occurrence of preceding expression	a+ 12+3

Regular Expressions

- Metacharacters (Characters with special meaning)

Metacharacter	Meaning	Example
?	Zero or one occurrence of preceding expression	-?[0-9]+

{}	● To specify already defined names
	● To specify number of occurrence
		{whitespace}
		1{2}3{4}5{6}

|	Or	a|b
()	Group series of regular expressions together	(ab|cd)+

Regular Expressions

- Metacharacters (Characters with special meaning)

Metacharacter	Meaning	Example
^	● If within [], then means except following characters ● Otherwise means start of line	[^ab] ^ab
$	End of line	124$
“”	Match anything literally	“^124$”
<<EOF>>	End of file	

Frequent Encounters

- yyin
- yylex()
- yywrap()
- yytext
- yylineno
- yyout

Start States

- One can declare start state in lex file
- By default, the start state is INITIAL

Thank You!