CSL302: Compiler Design

Tutorial on Lex

Vishwesh Jatala

Assistant Professor

Department of CSE

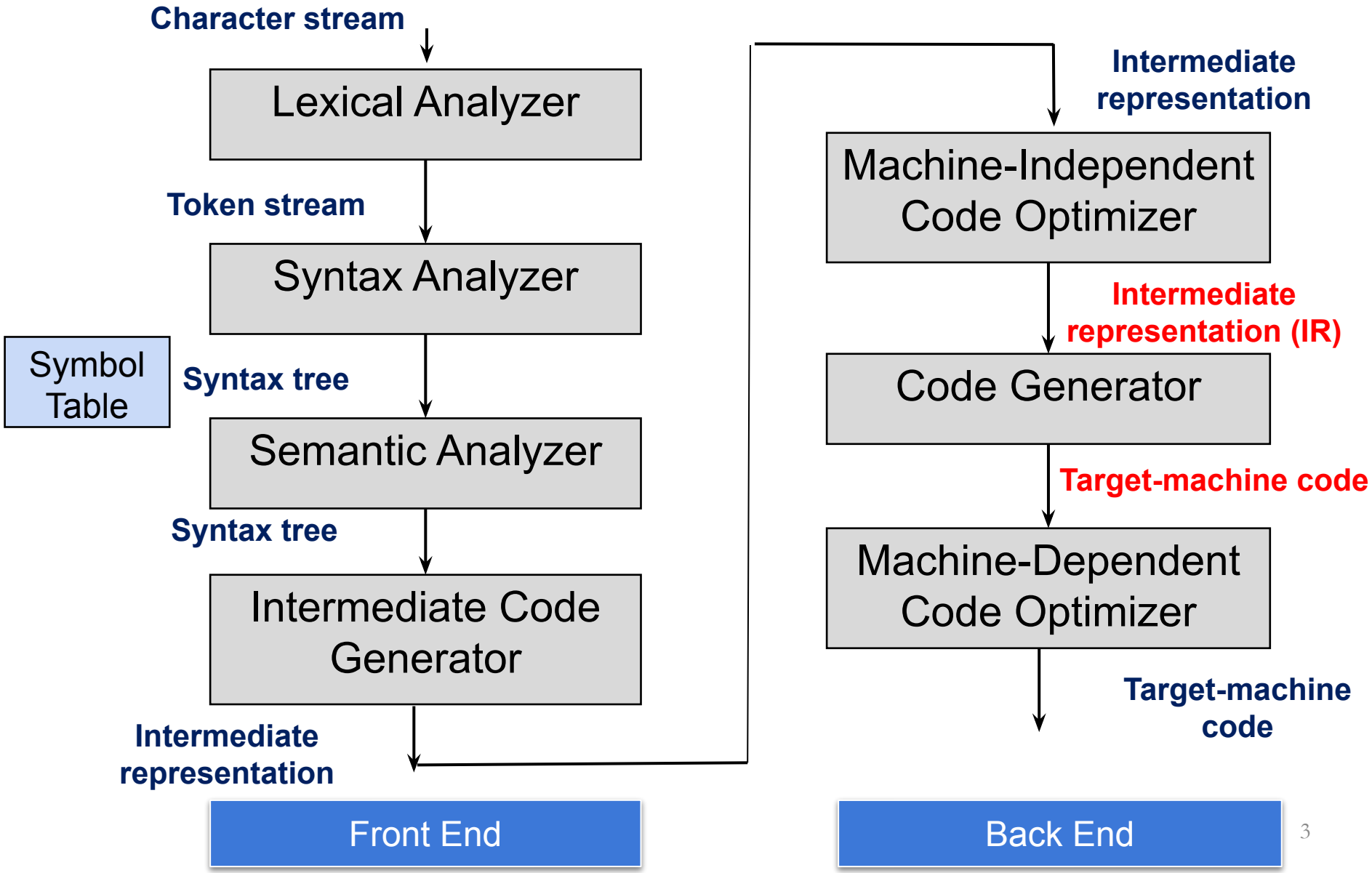
Indian Institute of Technology Bhilai

vishwesh@iitbhilai.ac.in

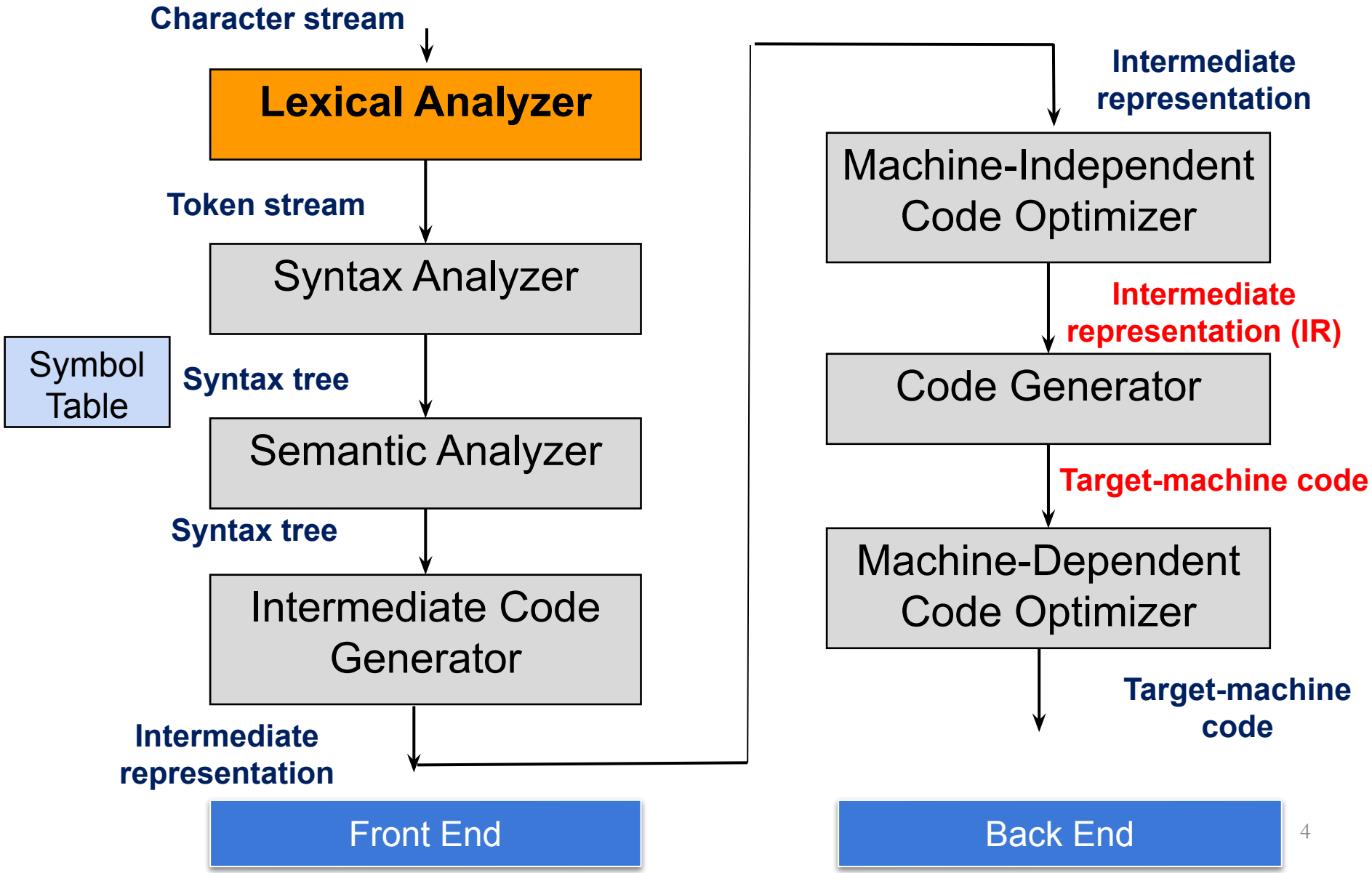
Acknowledgement

- References for today's slides
  - *Stanford University:*
    - *https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/*
  - *Lecture notes of Prof. Amey Karkare (IIT Kanpur) and Late Prof. Sanjeev K Aggarwal (IIT Kanpur)*
  - *Suggested textbook for the course*

Compiler Design



Compiler Design

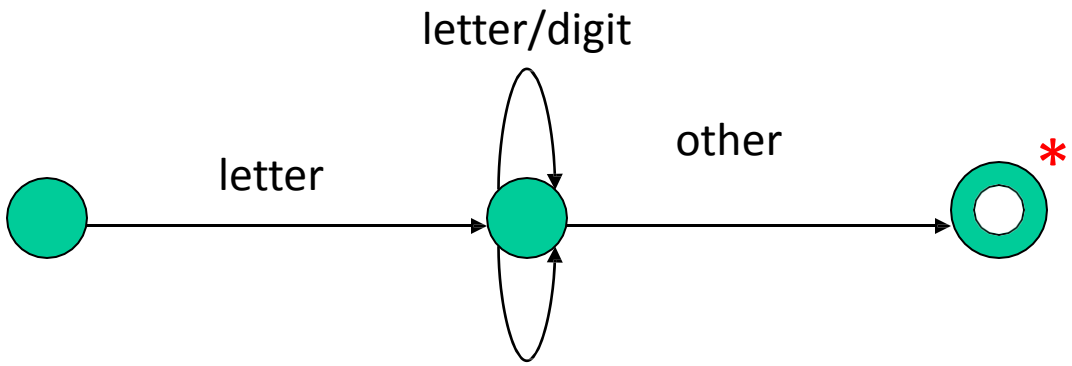


How to Recognize Tokens

- Consider

  id -> letter(letter|digit)*

Transition diagram for identifier



Approaches to implementation

- Use assembly language
  Most efficient but most difficult to implement

- Use high level languages like C
  Efficient but difficult to implement

Implementation of transition diagrams

```
Token nexttoken() {
  while(1) {
     switch (state) {
           ……
           case 10: c=nextchar();
            if(isletter(c))
            state=10;  elseif
            (isdigit(c)) state=10;
            else state=11;
            break;
           ……
           }
     }
}
```

Approaches to implementation

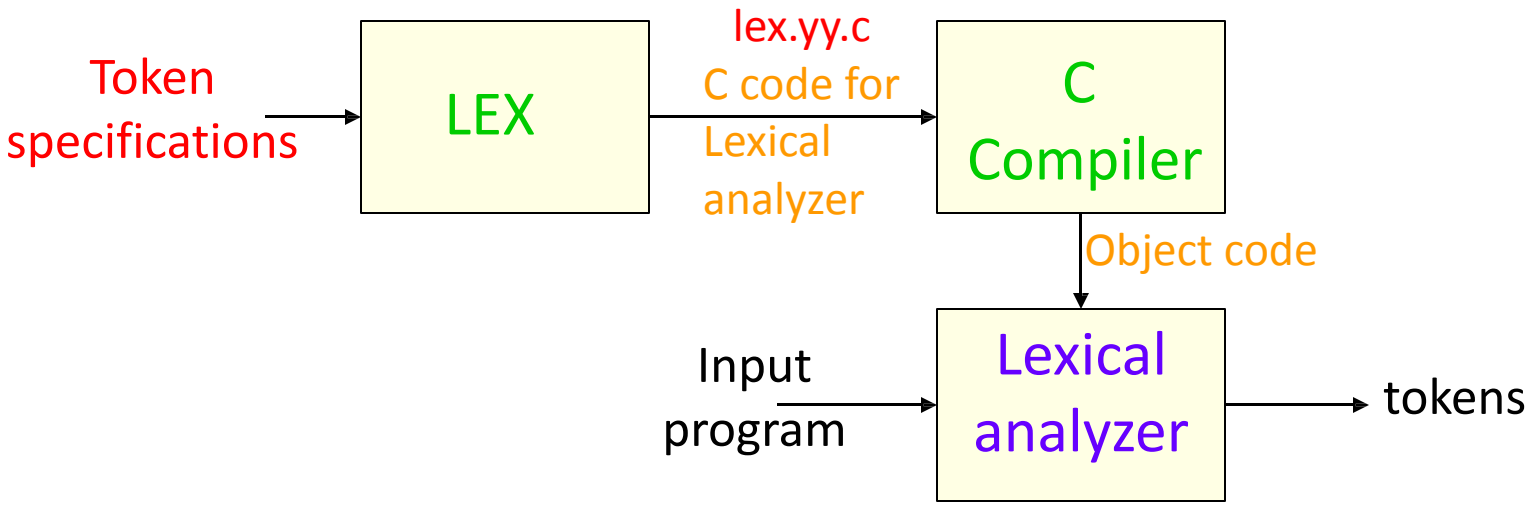
- Use assembly language
  Most efficient but most difficult to implement

- Use high level languages like C
  Efficient but difficult to implement

- Use tools like lex, flex
  Easy to implement

LEX: A lexical analyzer generator

- Lex is a tool that generates lexical analyzer.



Refer to LEX User's Manual

Token Specifications

- Regular definitions
  - Let r_i be a regular expression and d_i be a distinct name
  - Regular definition is a sequence of definitions of the form

    $d_1 \rightarrow r_1$
    $d_2 \rightarrow r_2$
    …..
    $d_n \rightarrow r_n$
  - Where each r_i is a regular expression

Lex: Program Structure

declarations

%%

transition rules

%%

auxiliary functions

Lex: Program Structure

- The LEX program has following structure:

```
%{
      C header files \* if required *\
      Definition section
%}

%%

Translation Rules

%%

Auxiliary functions
```

Structure of LEX Program (Contd..)

- The definition section defines macros and imports header files written in C.
- The rules section associates regular expression patterns with C statements.
- Auxiliary functions contains C statements and functions and contain code defined by the rules in the rules section

Regular Expressions

Table 1: Special Characters	
Pattern	**Matches**
.	any character except newline
\.	literal .
\n	newline
\t	tab
^	beginning of line
$	end of line

Regular Expressions

Table 2: Operators	
Pattern	**Matches**
?	zero or one copy of the preceding expression
*	zero or more copies of the preceding expression
+	one or more copies of the preceding expression
a|b	a or b (alternating)
(ab)+	one or more copies of ab (grouping)
abc	abc
abc*	ab abc abcc abccc ...
"abc*"	literal abc*
abc+	abc abcc abccc abcccc ...
a(bc)+	abc abcbc abcbcbc ...

LEX: A simple program

```
%{
    #include<stdio.h>
    #include<string.h>
    int i = 0;
%}
/* Rules Section*/
%%
    ([a-zA-Z0-9])*    {i++;} /* Rule for counting  number of words*/
    "\n"              {printf("%d\n", i); i = 0;}
%%
    int yywrap(void){return 1;}
    int main()
    {
        yylex();
       return 0;
    }
```

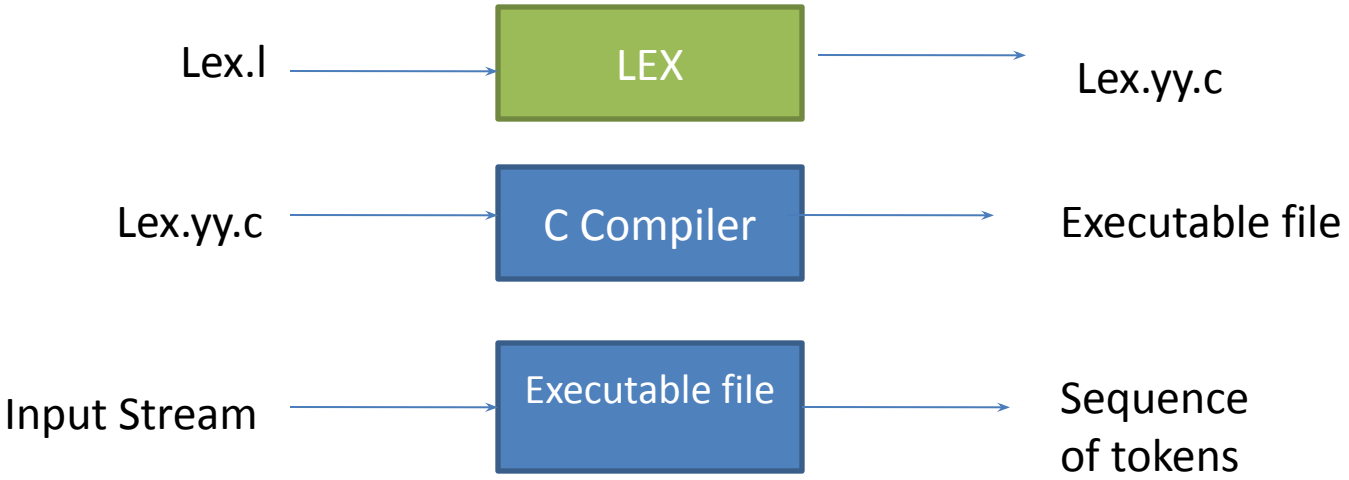
Important Functions

- **yywrap** is called by lex when input is exhausted. Return 1 if you are done or 0 if more processing is required.

- **yylex() :** The main point for lex. It reads the i/p stream and generates tokens also return 0 at the end of i/p stream. It is called to invoke the lexer.

Lex Compilation Overview

- The code is written in lex language. The extension .l is used for the lex program e.g filename.l
- The Lex compiler after compiling the lex source file i.e., filename.l always generates output as lex.yy.c



Flex

- Flex is a free and open-source software, which generates lexical analyzers.
- It is a tool for generating tokens
- It read the given input files.
- Flex generates as output a C source file, ‘lex.yy.c’, which defines a routine ‘yylex()’.
- When the executable is run, it analyzes its input for occurences of the regular expressions.
- Whenever it finds the pattern, it executes the corresponding C code.

Recognize Vowel and Cons

```
%{
    #include<stdio.h>
    int vowel=0;
    int cons=0;
%}
%%
"a"|"e"|"i"|"o"|"u"|"A"|"E"|"I"|"O"|"U" {printf("Vowel");}
[a-zA-z] {printf("Cons\n");}
%%
int yywrap() {return 1;}
main()
{
    printf("Enter String\n");
    yylex();
}
```

DEFINITION

RULES

Auxiliary functions