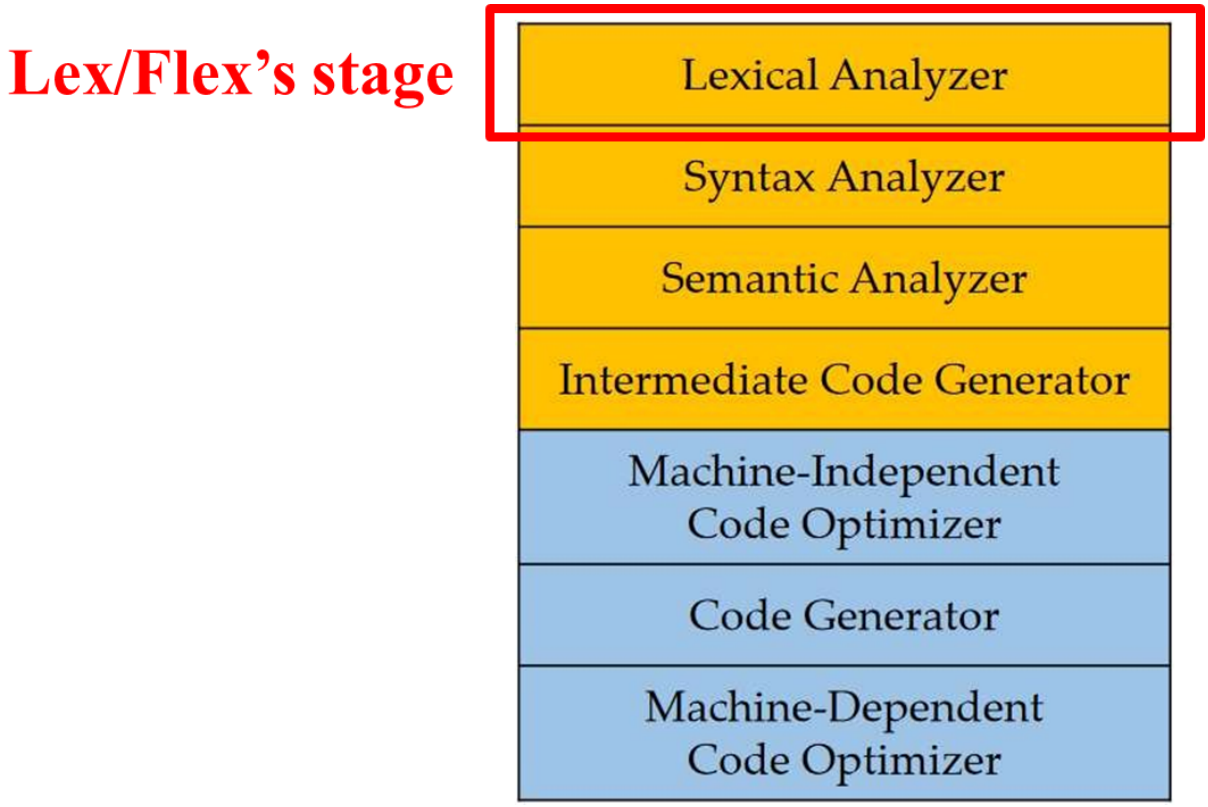
BUPT Compilers Lab2 2023Fall

Overview

Welcome to the lab class! In this course, you will be completing the programming assignments. Lab 2 involves the use of regular expressions and Flex, providing you with experience in handling text, which is crucial in compilers. Regular expressions and pattern matching are fundamental concepts in lexical analysis, laying the foundation for building the lexer of subsequent compilers.

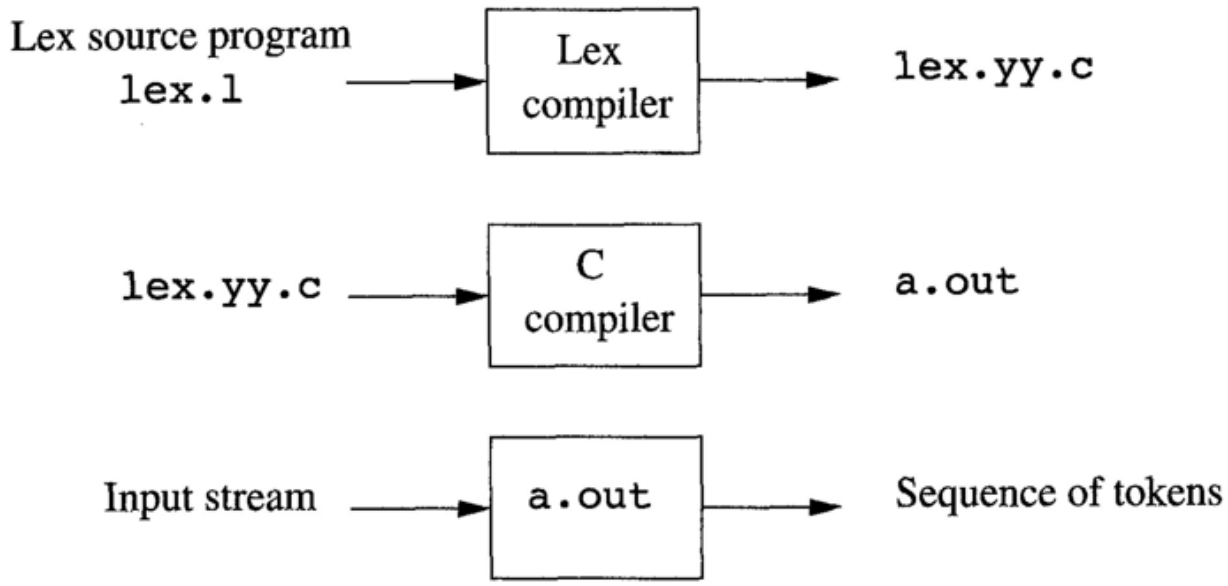
Lex/Flex Introduction



The Lexical-Analyzer Generator Lex

•Lex, or a more recent tool Flex, allows one to specify a lexical analyzer by specifying regexps to describe patterns for tokens

•Often used with Yacc/Bison to create the frontend of compiler



The lex.l describes the lexical analyzer in the Lex language.

The a.out is a working lexical analyzer.

Structure of Lex Programs

A Lex program has three sections separated by %%

- Declaration (声明)
  - Variables, manifest constants (e.g., token names)
  - Regular definitions
- Translation rules (转换规则) in the form “Pattern {Action}”
  - Each pattern (模式) is a regexp (may use the regular definitions of the declaration section)
  - Actions (动作) are fragments of code, typically in C, which are executed when the pattern is matched
- Auxiliary functions section (辅助函数)
  - Additional functions that can be used in the actions

Lex Program Example

```
%{
    /* definitions of manifest constants
    LT, LE, EQ, NE, GT, GE,
    IF, THEN, ELSE, ID, NUMBER, RELOP */
%}

/* regular definitions */
delim     [ \t\n]
ws        {delim}+
letter    [A-Za-z]
digit     [0-9]
id        {letter}({letter}|{digit})*
number    {digit}+(\.{digit}+)?(E[+-]?{digit}+)?

%%
```

Anything in between %{ and }% is copied directly to `lex.yy.c`.

In the example, there is only a comment, not real C code to define manifest constants

Regular definitions that can be used in translation rules

Section separator

```
{ws}      {/* no action and no return */}
if        {return(IF);}
then      {return(THEN);}
else      {return(ELSE);}
{id}      {yylval = (int) installID(); return(ID);}
{number}  {yylval = (int) installNum(); return(NUMBER);}
"<"       {yylval = LT; return(RELOP);}
"<="      {yylval = LE; return(RELOP);}
"="       {yylval = EQ; return(RELOP);}
"<>"      {yylval = NE; return(RELOP);}
">"       {yylval = GT; return(RELOP);}
">="      {yylval = GE; return(RELOP);}

%%
```

Continue to recognize other tokens

Return token name to the parser

Place the lexeme found in the symbol table

A global variable that stores a pointer to the symbol table entry for the lexeme. Can be used by the parser or a later component of the compiler.

- Everything in the auxiliary function section is copied directly to the file lex.yy.c
- Auxiliary functions may be used in actions in the translation rules

```
int installID() {/* function to install the lexeme, whose
                    first character is pointed to by yytext,
                    and whose length is yyleng, into the
                    symbol table and return a pointer
                    thereto */
}

int installNum() {/* similar to installID, but puts numer-
                     ical constants into a separate table */
}
```

Regular Expression

Pattern	Regex	Description
Character classes	`[0-9]`	This means alternation of the characters in the range listed (in this case: `0|1|2|3|4|5|6|7|8|9`). More than one range may be specified, e.g. `[0-9A-Za-z]` as well as specifying individual characters, as with `[aeiou0-9]`.
Character exclusion	`^`	The first character in a character class may be `^` to indicate the complement of the set of characters specified. For example, `[^0-9]` matches any non-digit character.
Arbitrary character	`.`	Matches any single character **except newline**.
Selection	`x|y`	Either `x` or `y` can be matched.
Single repetition	`x?`	0 or 1 occurrence of `x`.
Nonzero repetition	`x+`	`x` repeated one or more times; equivalent to `xx*`.
Specified repetition	`x{n,m}`	x repeated between n and m times.
Beginning of line	`^x`	Match `x` at beginning of line only.
End of line	`x$`	Match `x` at end of line only.
Context-sensitivity	`ab/cd`	Match `ab` but only when followed by `cd`. The lookahead characters are left in the input stream to be read for the next token.
Literal strings	`"x"`	This means `x` even if `x` would normally have special meaning. Thus, `"x*"` may be used to match `x` followed by an asterisk. You can turn off the special meaning of just one character by preceding it with a backslash, .e.g. `\.` matches exactly the period character and nothing more.
Definitions	`{name}`	Replace with the earlier defined pattern called `name`. This kind of substitution allows you to reuse pattern pieces and define more readable patterns

For thoes unfamiliar with regex:https://regex101.com/

Coding

> **Running the wc program example**
>
> Make the wc target, run it and compare with system's wc.(see the code under lab2/wc)
>
> The commands you may use:
>
> ```
> make wc
> ./wc.out inferno3.txt
> wc inferno3.txt
> ```
>
> 1. Provide a screenshot of the operation results as follows.
> 2. Explain why the output of running this wc program is different from the wc command on Linux systems.

> **Flex Exercises - identifiers**
>
> Make the identifiers target and run it. You will find that the token and line number do not match in the output. You need to solve it with Flex in the lab session(see the code under lab2/identifiers).
>
> The commands you may use:
>
> ```
> make idcount
> ./idcount.out ./test.c
> ```
>
> 1. Provide a screenshot of the operation results.
> 2. Explain the logic or principles of key parts of your code.

> **Flex Exercise - ipaddr**
>
> Validate IP Address. Solve it with Flex in the lab session (see the code under lab2/ipaddr) and use ip_test.py to verify the results. Try to pass them!
>
> The commands you may use:
>
> ```
> make ipaddr
> python3 ip_test.py
> ```
>
> 1. Provide a screenshot of the operation results.
> 2. Explain the logic or principles of key parts of your code.

Report

Additionally, you will need to submit a pdf report(name_studentID.pdf) documenting your work for assignment. Please carefully follow the instructions outlined below:

1. Academic Integrity: Plagiarism or any form of cheating is strictly prohibited. Your work should be original, and any external sources should be appropriately cited.
2. Programming Assignments: Feel free to ask questions and seek assistance from the teaching assistant if needed.
3. Report:
    - Pdf type
    - Naming like **name_studentID.pdf**
    - Include any relevant diagrams, charts, or screenshots to enhance your explanations.
    - Make sure your report is well-structured, with appropriate headings and subheadings.
    - Place under lab2 folder.
4. Submission Guidelines:

- Include your **name, student ID, class number** and **container ID** in the report's header. For Docker on Windows systems, you can view the container numbers in the containers of the Docker Desktop.
- Commit the compressed package of the lab2 folder（**lab2.zip**）.

5. Deadline:

   October 8, 2023, 23:59

6. Submission Platform:

   Teaching cloud platform

Report format

Name: xxx

Student ID: xxxxxxxxxx

Class Number: xxxxxxxxxx

Container ID: 04c56e0ced1b3944f2eeee027840f5210f54efecc49b3377166c9a7a32ff119

Your content