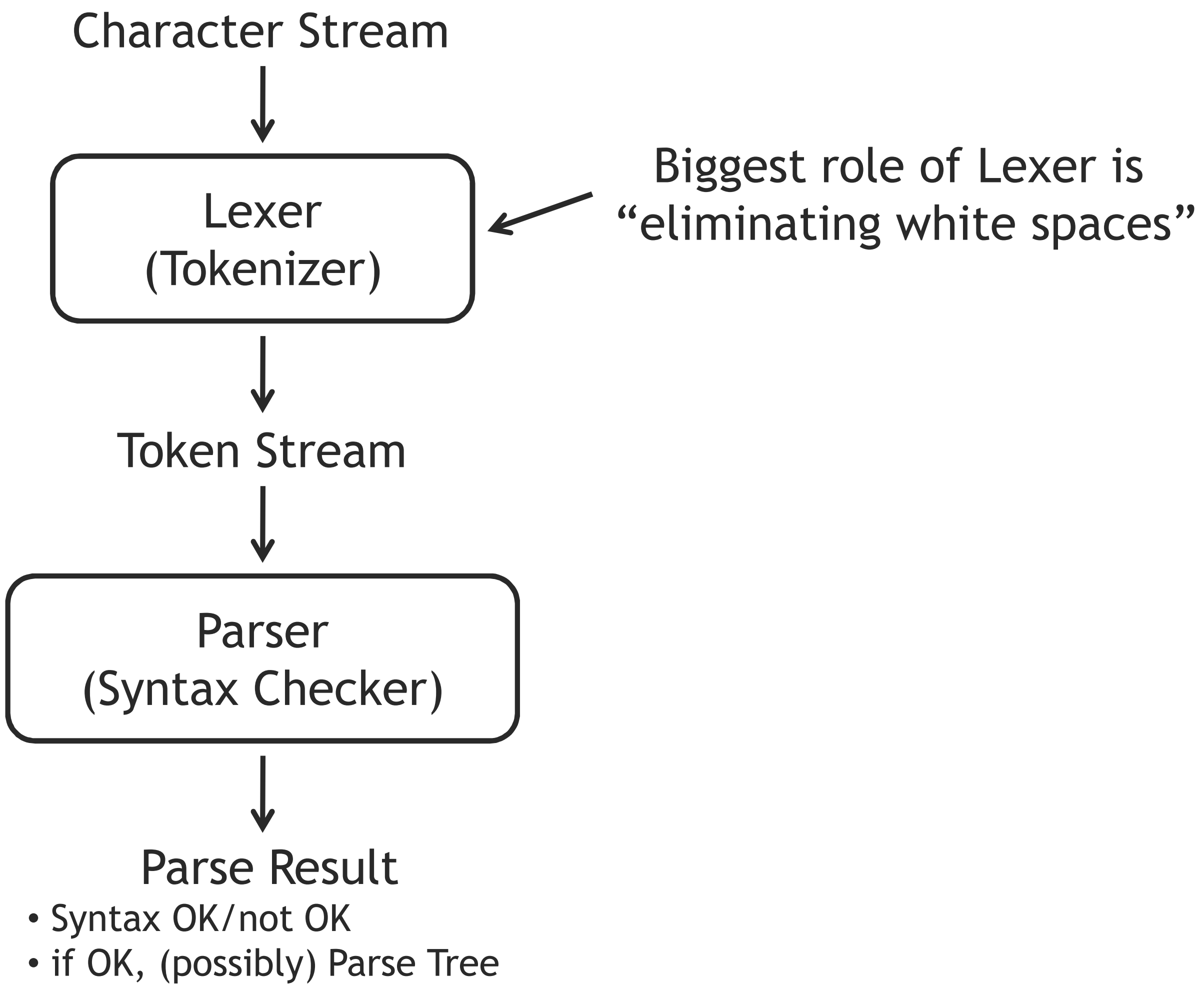
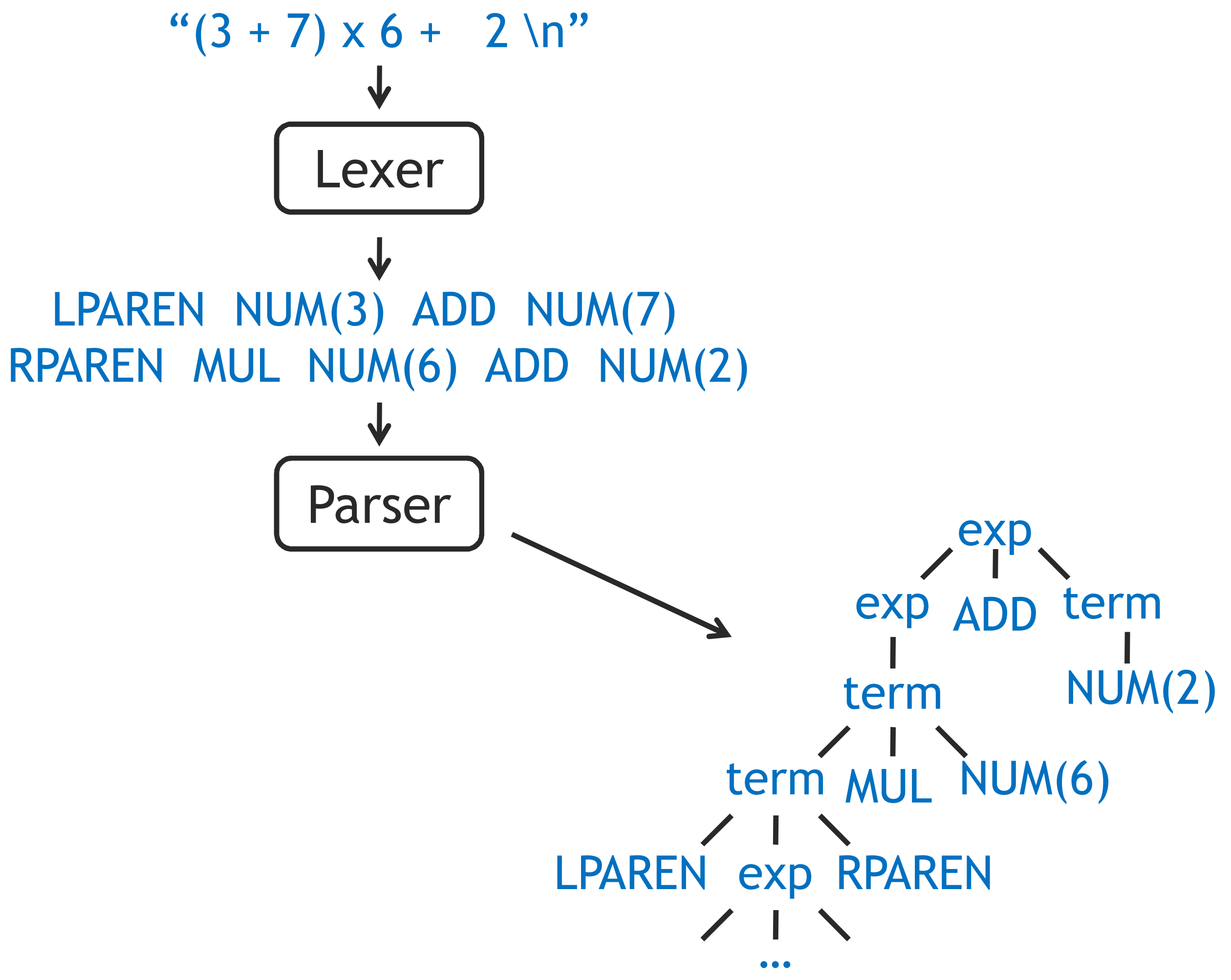
Introduction to Lex & Yacc

Presented in CS322 Course
by T.A. Hyunik Na
CS Dep. at KAIST

What is Lexer & Parser ?



Lexer & Parser: An Example

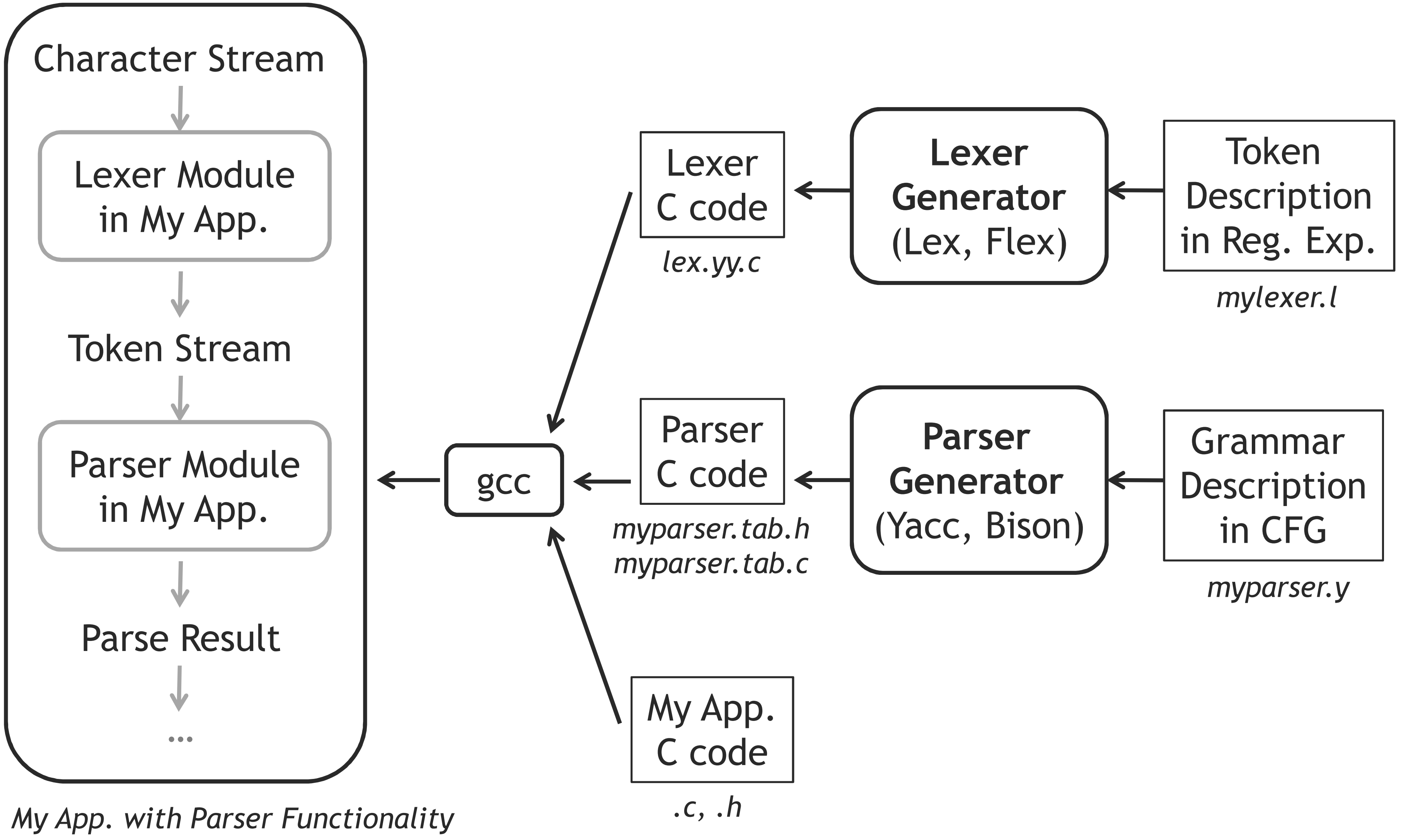


Lexer & Parser Generators

- Should we rewrite lexer(parser) for every lexical convention(grammar) ?
  - Fortunately, No

- We have generators
  - Lexer Generator: Lex, Flex
  - Parser Generator: Yacc, Bison

Lexer & Parser Generators



Simplest example of lex input file

<cat.l>

```
%%

%%
```

Hidden default rule:

“.|\n ECHO;”
(echo any input character)

<lexer generation and the result>

```
[hina@cc1 test]$ ls
cat.l
[hina@cc1 test]$ flex cat.l
[hina@cc1 test]$ ls
cat.l  lex.yy.c
[hina@cc1 test]$ gcc lex.yy.c -lfl
[hina@cc1 test]$ ls
a.out*  cat.l  lex.yy.c
[hina@cc1 test]$ ./a.out
abcde
abcde
hijkl
hijkl
^C
[hina@cc1 test]$
```

Format of Lex Input File

```
%{
#include <stdio.h>
#include <stdlib.h>
void mycode();
%}
```

C code : Copied to output file

```
LETTER   [A-Za-z]
DIGIT    [0-9]
%option main
```

Name definitions, Options

```
%%

({LETTER})({LETTER}|{DIGIT})*   { printf("WORD\n"); }
({DIGIT})+                      { printf("NUMBER\n"); mycode(); }

%%
```

Rules:
*(Pattern + Action)**

```
void mycode() {
    printf("lex example: (%d)\n“, atoi(yytext) );
}
```

C code : Copied to output file

Lex file example 2 : verbs.l

<verbs.l>

```
%%

[\t ]+      /* no action, do nothing */    ;
is |
am |
are |
were |
do |
does |
did |
have |
had |
go           {printf("%s: is a verb\n", yytext);}
[a-zA-Z]+    {printf("%s: is not a verb\n“,yytext);}
.|\n         ECHO;

%%

main()
{ yylex(); }
```

<execution result>

```
[hina@cc1 test]$ ./a.out
did I have a fun ?
did: is a verb
I: is not a verb
have: is a verb
a: is not a verb
fun: is not a verb
?
^C
[hina@cc1 test]$
```

- Pattern matching
  - Longest match
  - Only once
  - Topmost among the candidates

Yacc - Yet Another Compiler-Compiler

- Input file format: Similar to Lex'

```
%{
User Codes
%}
Declarations – tokens, token types, options, …
%%
Grammar Rules (extended BNF Form)
%%
User Codes
```

- Output:
  - <myparser>.tab.h, <myparser>.tab.c

Yacc & Lex Cooperation

```
/* Definitions */
%{
#include <stdio.h>
%}
%token NUMBER PLUS MULT

%%
/* Rules */
expr: NUMBER { $$ = $1; }
     | expr PLUS expr { $$ = $1 + $3; }
     | expr MULT expr { $$ = $1 * $3; }
     ;

%%
/* User Code */
void yyerror(char *s)
{ printf("%s₩n", s); }
```

<Yacc file – calc.y>

```
/* Definitions */
%{
#include <stdlib.h>
#include "calc.tab.h"
%}
NUMBER [0-9]+

%%
/* Rules */
{NUMBER} { yylval = atoi(yytext);
           return NUMBER; }
"+"      { return PLUS; }
"*"      { return MULT; }

%%
/* User Code */
```

<Lex file – calc.l>

Further Readings

- This is just a 30 minutes introduction to Lex & Yacc
- Read further resources for details
  - example on the course web board (to be uploaded)
  - man pages for flex & bison
  - documents on the web
    - http://flex.sourceforge.net/
    - http://www.gnu.org/software/bison/manual/index.html
  - books
    - Flex & Bison by John Levine and Levine John
- Have fun with Lex & Yacc !