Introduction to Flex and Bison

Samuel Malec

Overview

- Motivation

Overview

- Motivation
- Lexical analysis using Flex

Overview

- Motivation
- Lexical analysis using Flex
- Syntactical analysis using Bison

Overview

- Motivation
- Lexical analysis using Flex
- Syntactical analysis using Bison
- Live Demo

Motivation

- Source code is just **text**

Motivation

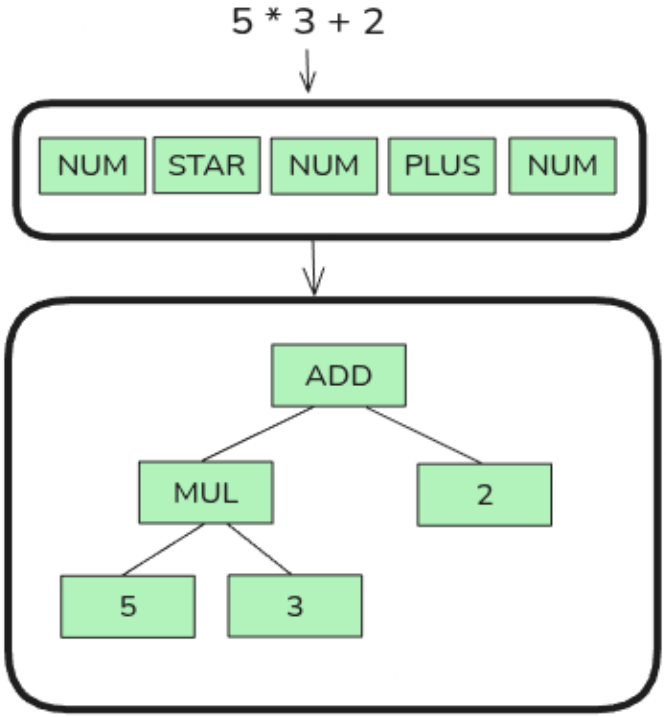
- Source code is just **text**
- Compiler needs to recognize its **structure**

Motivation

- Source code is just **text**
- Compiler needs to recognize its **structure**
- **Lexers** and **parsers** serve this purpose

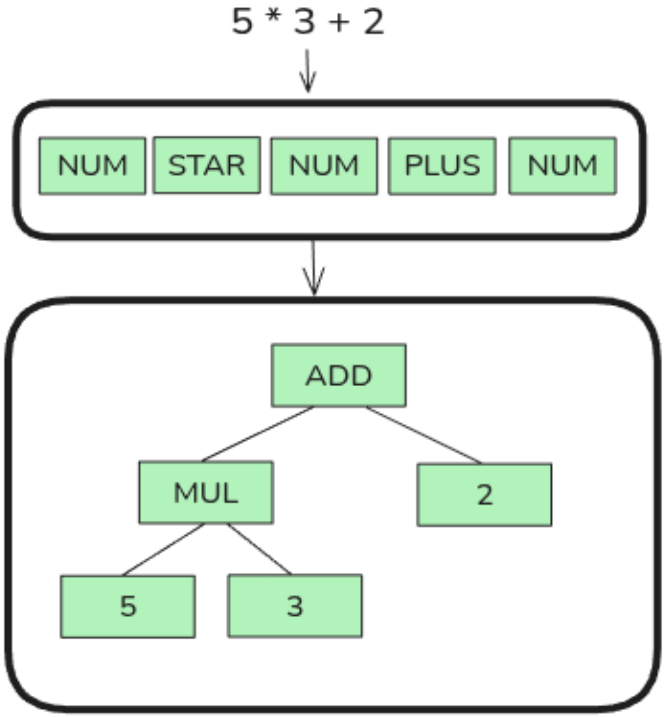
Motivation

- Source code is just **text**
- Compiler needs to recognize its **structure**
- **Lexers** and **parsers** serve this purpose



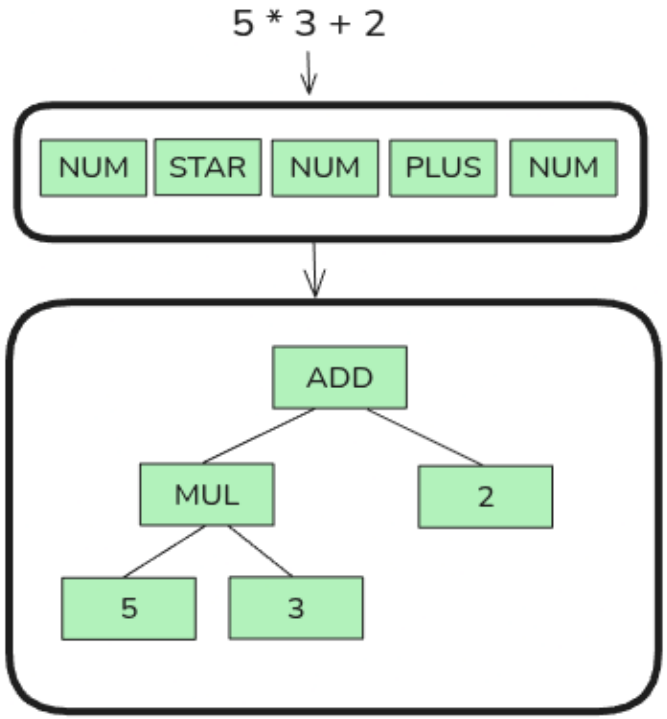
Motivation

- Source code is just **text**
- Compiler needs to recognize its **structure**
- **Lexers** and **parsers** serve this purpose
- Repetitive and error-prone code



Motivation

- Source code is just **text**
- Compiler needs to recognize its **structure**
- **Lexers** and **parsers** serve this purpose
- Repetitive and error-prone code
- We can use **automated tools** !



Flex

- Fast **Lex**ical Analyzer (Generator)

Flex

- **F**ast **Lex**ical Analyzer (Generator)
- Open-source tool for **generating** lexical analyzers

Flex

- **F**ast **Lex**ical Analyzer (Generator)
- Open-source tool for **generating** lexical analyzers
- Users define rules for tokens using **regular expressions**

Flex

- **Fast Lex**ical Analyzer (Generator)
- Open-source tool for **generating** lexical analyzers
- Users define rules for tokens using **regular expressions**
- Flex produces C functions that return tokens

Flex

- **Fast Lex**ical Analyzer (Generator)
- Open-source tool for **generating** lexical analyzers
- Users define rules for tokens using **regular expressions**
- Flex produces C functions that return tokens

```
[0-9]+      { return NUMBER; }
if          { return IF; }
[a-zA-Z]+   { return IDENT; }
[ \t\r\n]+  // skip whitespace
```

Flex

- **Fast Lex**ical Analyzer (Generator)
- Open-source tool for **generating** lexical analyzers
- Users define rules for tokens using **regular expressions**
- Flex produces C functions that return tokens
- Conflicts are resolved using **longest match** and **rule ordering**

```
[0-9]+      { return NUMBER; }
if          { return IF; }
[a-zA-Z]+   { return IDENT; }
[ \t\r\n]+  // skip whitespace
```

Bison

- GNU tool for generating **parsers**

Bison

- GNU tool for generating **parsers**
- Users define **grammar rules** and optional **semantic actions**

Bison

- GNU tool for generating **parsers**
- Users define **grammar rules** and optional **semantic actions**
- Bison produces C-code that implements a **LALR(1) parser** for the defined grammar

Bison

- GNU tool for generating **parsers**
- Users define **grammar rules** and optional **semantic actions**
- Bison produces C-code that implements a **LALR(1) parser** for the defined grammar

```
%token NUM    // token declaration
%%
// nt : rule {action}
expr : expr '+' term   { $$ = $1 + $3; }
     | expr '-' term   { $$ = $1 - $3; }
     | term;
term : NUM;
%%
```

Live Demo

- We can use Flex and Bison together to create the **frontend** of a compiler
- Let see how to do it in a simple language

Calc

- A minimal integer calculator

Calc

- A minimal integer calculator
- Supports variables, assignments, arithmetic, parentheses, and a print(expr) statement.

Calc

- A minimal integer calculator
- Supports variables, assignments, arithmetic, parentheses, and a print(expr) statement.
- **Expressions**: '+ - / *' with standard precedence

Calc

- A minimal integer calculator
- Supports variables, assignments, arithmetic, parentheses, and a print(expr) statement.
- **Expressions**: '+ - / *' with standard precedence

```
a = 3;
b = 4;
print(a + b * 2);
print((a + b) * 2);
```

Example Calc program

Calc

- A minimal integer calculator
- Supports variables, assignments, arithmetic, parentheses, and a print(expr) statement.
- **Expressions**: '+ - / *' with standard precedence
- Let's look at the implementation!

```
a = 3;
b = 4;
print(a + b * 2);
print((a + b) * 2);
```

Example Calc program