

Compiler Construction

Lexical Analysis

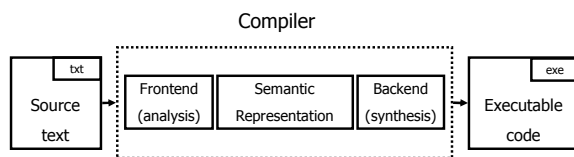
Rina Zviel-Girshin and Ohad Shacham
School of Computer Science
Tel-Aviv University

Administration

- Project Teams
 - Send me your group
 - Send me an email if you can't find a team
- First PA – for two weeks
- There are no recitations next week
 - November 26 – Schreiber 07
 - 9:00 – 10:00
 - 13:00 – 14:00
 - November 27 – Schreiber 07
 - 10:00 – 11:00

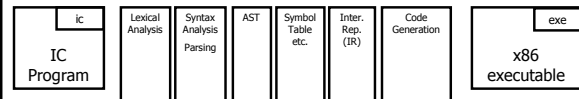
2

Generic compiler structure



3

IC compiler Compiler



4

Lexical Analysis

- converts characters to tokens

```
class Quicksort {
  int[] a;
  int partition(int low, int high)
  {
    int pivot = a[low];
    ...
  }
}
1: CLASS
1: CLASS_ID(Quicksort)
1: LCBR
2: INT
2: LB
2: RB
2: ID(a)
2: SEMI
...
```

5

Lexical Analysis

- Tokens
 - ID – _size, _num
 - Num – 7, 5, 9, 4926209837
 - COMMA – ,
 - SEMI – ;
 - ...
- Non tokens
 - Comment – //
 - Whitespace
 - Macro
 - ...

6

Problem

- Input
 - Program text
 - Tokens specification
- Output
 - Sequence of tokens

```
class Quicksort {
    int[] a;
    int partition(int low, int high)
    {
        int pivot = a[low];
        ...
    }
}
1: CLASS
1: CLASS_ID(Quicksort)
1: LCBR
2: INT
2: LB
2: RB
2: ID(a)
2: SEMI
...
```

7

Solution

- Write a lexical analyzer

```
Token nextToken()
{
    char c;
    loop: c = getchar();
    switch (c) {
        case '\n': goto loop;
        case ';': return SemiColumn;
        case '+': c = getchar();
            switch (c) {
                case '+': return PlusPlus;
                case '=': return PlusEqual;
                default: ungetc(c);
                    return Plus;
            }
        case '<':
        case 'w':
            ...
    }
}
```

8

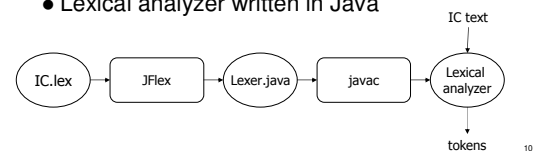
Solution's Problem

- A lot of work
- Corner cases
- Error prone
- Hard to debug
- Exhausting
- Boring
- Hard to reuse
- Switch parser's code between people
-
-

9

JFlex

- Off the shelf lexical analysis generator
- Input
 - scanner specification file
- Output
 - Lexical analyzer written in Java



10

JFlex

- Simple
- Good for reuse
- Easy to understand
- Many developers and users debugged the generators

```
"+" { return new symbol (sym.PLUS); }
"boolean" { return new symbol (sym.BOOLEAN); }
"int" { return new symbol (sym.INT); }
>null" { return new symbol (sym.NULL); }
"while" { return new symbol (sym.WHILE); }
"=" { return new symbol (sym.ASSIGN); }
...
```

11

JFlex Spec File

- User code
 - Copied directly to Java file

Possible source of javac errors down the road

- %%
 - JFlex directives
 - Define macros, state names

```
DIGIT = [0-9]
LETTER = [a-zA-Z]
YYINITIAL
```

- %%
 - Lexical analysis rules
 - How to break input to tokens
 - Action when token matched

```
{LETTER}
{LETTER}{DIGIT}*
```

12

User code

```
package IC.Parser;
import IC.Parser.Token;

...
any scanner-helper Java code
...
```

13

JFlex Directives

- Control JFlex internals
 - `%line` switches line counting on
 - `%char` switches character counting on
 - `%class class-name` changes default name
 - `%cup` CUP compatibility mode
 - `%type token-class-name`
 - `%public` Makes generated class public (package by default)
 - `%function read-token-method`
 - `%scanerror exception-type-name`

14

JFlex Directives

- State definitions
 - `%state state-name`
 - `%state STRING`
- Macro definitions
 - `macro-name = regex`

15

Regular Expression

<code>r\$</code>	match reg. exp. <i>r</i> at end of a line
<code>.</code>	any character except the newline
<code>"..."</code>	string
<code>{name}</code>	macro expansion
<code>*</code>	zero or more repetitions
<code>+</code>	one or more repetitions
<code>?</code>	zero or one repetitions
<code>(...)</code>	grouping within regular expressions
<code>a b</code>	match <i>a</i> or <i>b</i>
<code>[...]</code>	class of characters - any <u>one</u> character enclosed in brackets
<code>a-b</code>	range of characters
<code>[^...]</code>	negated class - any one not enclosed in brackets

16

Example macros

```
ALPHA=[A-Za-z_]
DIGIT=[0-9]
ALPHA_NUMERIC={ALPHA}|{DIGIT}
IDENT={ALPHA}({ALPHA_NUMERIC})*
NUMBER=({DIGIT})+
NUMBER=[0-9]+
```

17

Rules

`[states] regexp {action as Java code}`

• Priorities

- Longest match
- Order in the lex file
- Rules should match all inputs!!!

18

Rules Examples

```
<YYINITIAL> {DIGIT}+ {
  return new Symbol(sym.NUMBER, yytext(), yyline);
}

<YYINITIAL> "-" {
  return new Symbol(sym.MINUS, yytext(), yyline);
}

<YYINITIAL> [a-zA-Z] ([a-zA-Z0-9]) * {
  return new Symbol(sym.ID, yytext(), yyline);
}
```

19

Rules – Action

- Action
 - Java code
 - Can use special methods and vars
 - yyline
 - yytext()
 - Returns a token for a token
 - Eats chars for non tokens

20

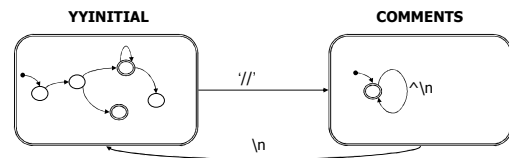
Rules – State

- State
 - Which regexp should be evaluated?
 - yybegin(stateX)
 - jumps to stateX
 - YYINITIAL
 - JFlex's initial state

21

Rules – State

```
<YYINITIAL> "/" { yybegin(COMMENTS); }
<COMMENTS> [^\n] { }
<COMMENTS> [\n] { yybegin(YYINITIAL); }
```



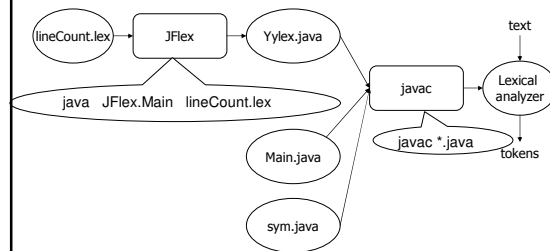
22

Lines Count Example

```
import java_cup.runtime.Symbol;
%%
%cup
%{
  private int lineCounter = 0;
}%
%eofval{
  System.out.println("line number=" + lineCounter);
  return new Symbol(sym.EOF);
}%eofval
NEWLINE=\n
%%
<YYINITIAL>{NEWLINE} {
  lineCounter++;
}
<YYINITIAL>{^[NEWLINE]} { }
```

23

Lines Count Example



JFlex and JavaCup must be on CLASSPATH

24

Test Bed

```
import java.io.*;

public class Main {
    public static void main(String[] args) {
        Symbol currToken;
        try {
            FileReader txtFile = new FileReader(args[0]);
            Yylex scanner = new Yylex(txtFile);
            do {
                currToken = scanner.next_token();
                // do something with currToken
            } while (currToken.sym != sym.EOF);
        } catch (Exception e) {
            throw new RuntimeException("IO Error (brutal exit)" +
                e.toString());
        }
    }
}
```

25

Common Pitfalls

- Classpath
- Path to executable
- Define environment variables
 - JAVA_HOME
 - CLASSPATH

26

Programming Assignment 1

- Implement a scanner for IC
- **class Token**
 - At least - line, id, value
 - Should extend `java_cup.runtime.Symbol`
 - Numeric token ids in `sym.java`
 - Will be later generated by JavaCup
- **class Compiler**
 - Testbed - calls scanner to print list of tokens
- **class LexicalError**
 - Caught by Compiler
- **Don't forget to generate scanner and recompile Java sources when you change the spec**
- You need to download and install **both** JFlex and JavaCup

27

sym.java

```
public class sym {
    public static final int EOF = 0;
    public static final int ID = 1;
    ...
}
```

- Defines symbol constant ids
- Communicate between parser and scanner
- Actual values don't matter
 - Unique value for each token
- Will be generated by cup in PA2

28

Token class

```
import java_cup.runtime.Symbol;

public class Token extends Symbol {
    public int getId() {...}
    public Object getValue() {...}
    public int getLine() {...}
    ...
}
```

29

JFlex directives to use

<code>%cup</code>	(integrate with cup)
<code>%line</code>	(count lines)
<code>%type Token</code>	(pass type Token)
<code>%class Lexer</code>	(gen. scanner class)

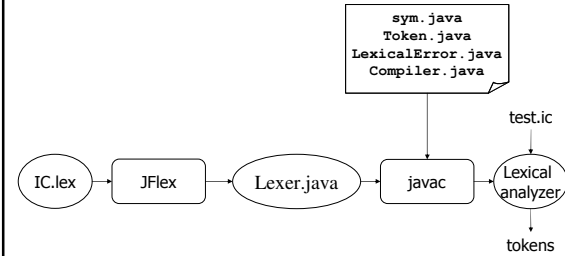
30

%cup

- %implements java_cup.runtime.Scanner
 - Lex class implements java_cup.runtime.Scanner
- %function next_token
 - Returns the next token
- %type java_cup.runtime.Symbol
 - Return token Class

31

Structure



32

Directions

- Download Java
- Download JFlex
- Download JavaCup
- Put JFlex and JavaCup in classpath
- Eclipse
 - Use ant build.xml
 - Import jflex and javacup
- Apache Ant

33

Directions

- Use skeleton from the website
- Read Assignment
- Use Forum

34