Proyecto 1

José Pablo Murillo Vargas, Nicol Morice Sandí

Instituto Técnológico de Costa Rica

Escuela de Ingeniería en Computación

Compiladores e Intérpretes

19-09-2017

Explicación del Scanner

El scanning, o análisis léxico es el proceso por el cuál se toma el archivo fuente y se intenta de romper en pedacitos significativos llamados tokens.

Los scanners funcionan al buscar patronos de carácteres dentro del fuente. Una forma de expresar estos patronos es por medio de las **expresiones regulares**. Por ejemplo el patrón [1-9][0-9]* sería un patrón útil para encontrar todos los números enteros dentro del archivo fuente.

Explicación de Flex

Flex (Fast lexical analyzer generator) es un programa que genera los analizadores léxicos o scanners. Su propósito era mejorar las deficiencias y pulgas de Lex.

Flex fue escrito originalmente por Jef Poskanzer y luego fué mejorado por Vern Paxson y Van Jacobson.

Un programa en flex es usar un conjunto de expresiones regulares y escribir que acciones debe tomar cuándo encuentre uno de estos en el fuente.

Flex también traduce todas las expresiones regulares en una forma interna eficiente que le permite hacer match simultáneo de todos los patrones con el archivo fuente.

```
typedef unsigned short U ; U ( main ) [ 32768 ] , n , r [ 8 ] ; int
__attribute__ ( (

* / char * ) main + ( x ) )

- 4 ? v += 2 , C ( i - 6 ? v - 2 : v + * C ( v - 2 ) ) : C ( v -= 2 )
: &v )
```

- Keywords
- Identifiers
- Numbers
- String-literal
- Operators
- Punctuators

```
constructor ) ) U ( x ) ( ) { for ( ; ; * r += 2 , * r += ! n ? _exit
( write ( 2 , "llleg"

"al ins" "truction ;-" "(
" , 24 ) ) , 0 : n >> 8 == 001 ? ( signed char
```

- Keywords
- Identifiers
- Numbers
- String-literal
- Operators
- Punctuators

```
) n * 2 : 548 == n >> 6 &&usleep ( 10
) + n %64 == 4 ? 0 * write ( r [ 7 ] , C (
* C ( * r ) ) , * C ( * r + 2 ) ) + 4 : n >> 9
== 63 &&--r [ 7 - n / 64 %8 ] ? n %+ 64 * -
```

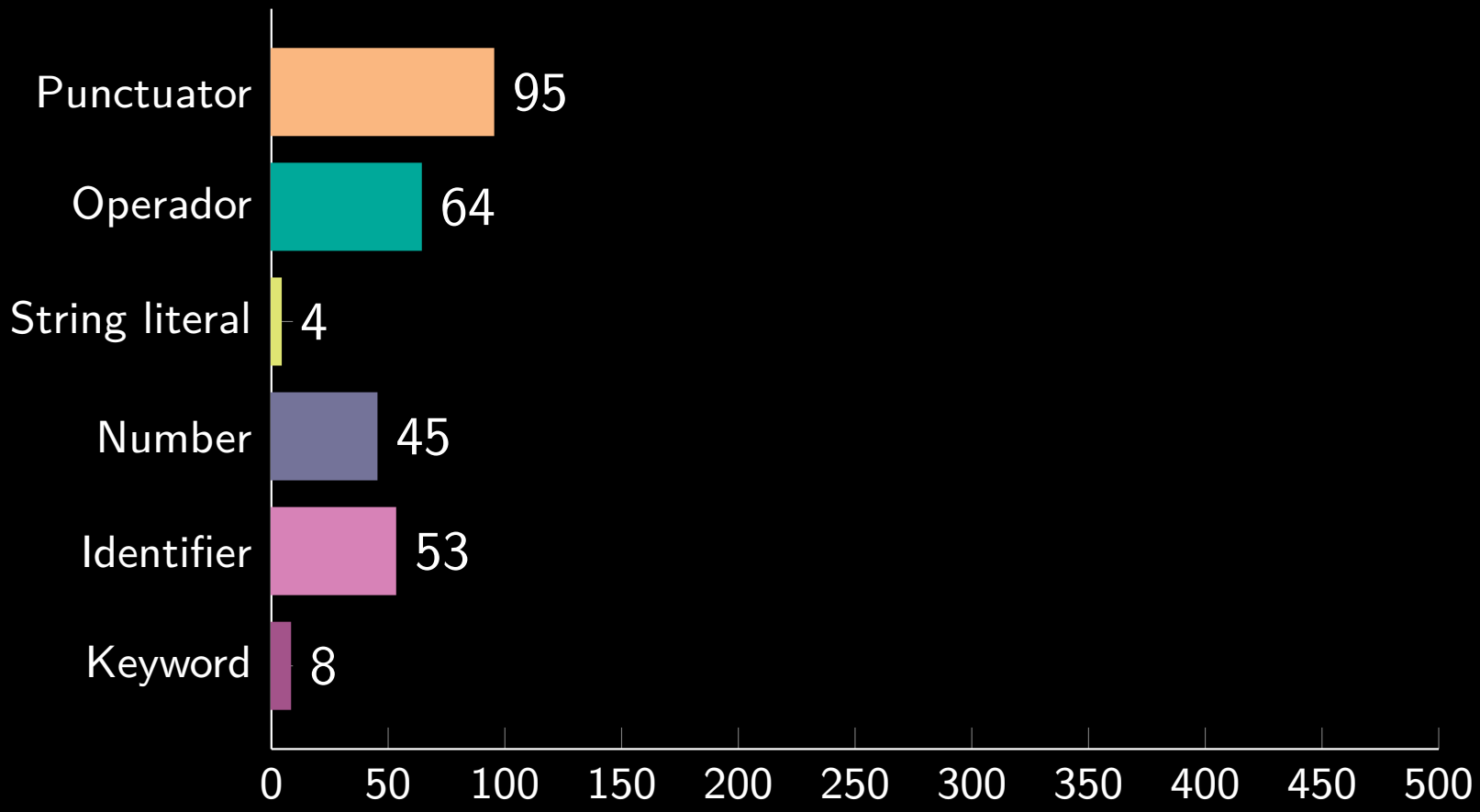
- Keywords
- Identifiers
- Numbers
- String-literal
- Operators
- Punctuators

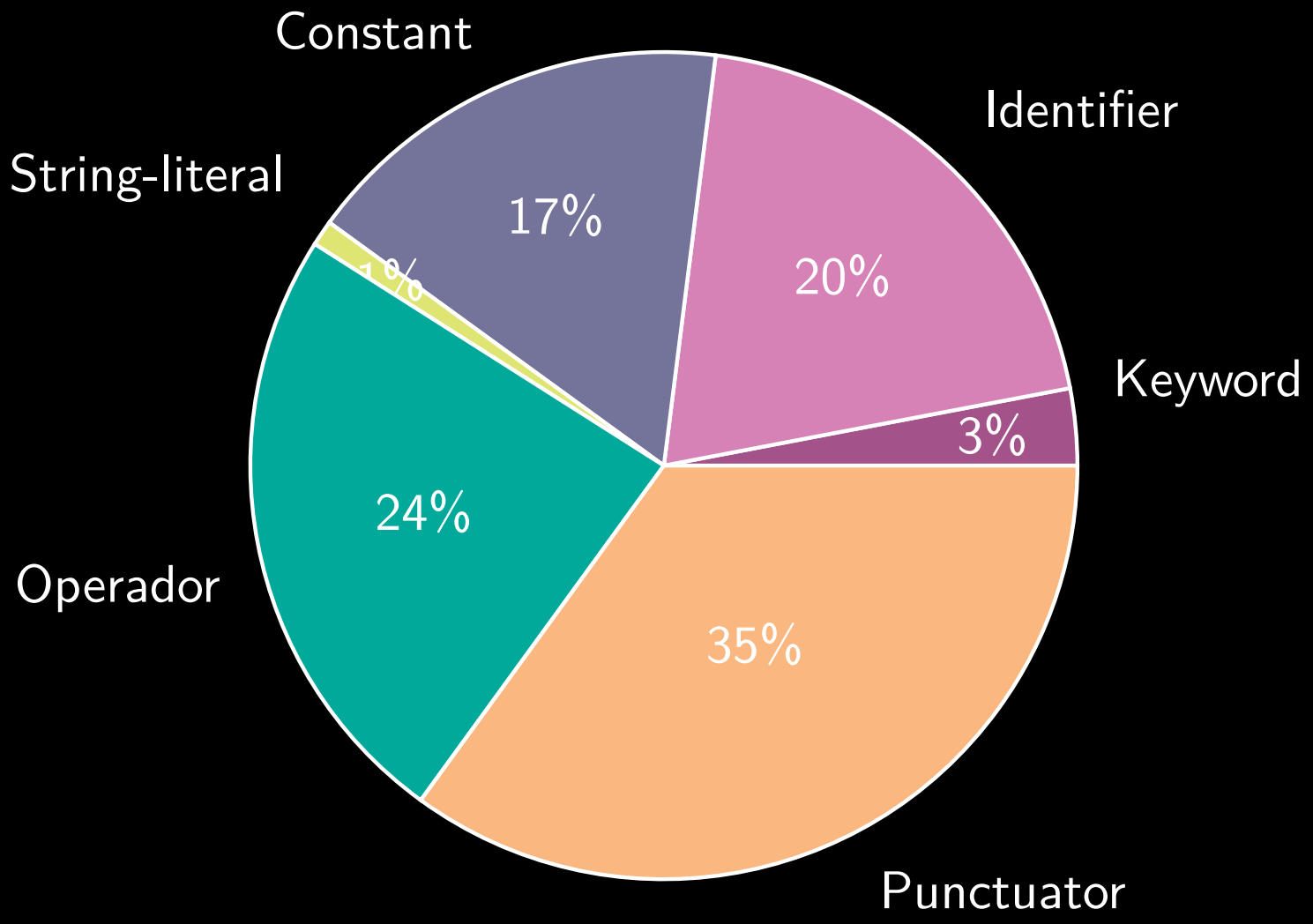
```
2 : 0 , n >> 6 == 47 ? * R ( 0 ) : n >> 12 == 1 ?

* R ( 0 ) = * R ( + 6 ) : n >> 12 == + 14 ? *

R ( 0 ) -= * R ( 2 * 3 ) : 0 ) n = * C ( * r ) ; }
```

- Keywords
- Identifiers
- Numbers
- String-literal
- Operators
- Punctuators


Punctuator
95
Operador
64
String literal
4
Number
45
Identifier
53
Keyword
8
0
50
100
150
200
250
300
350
400
450
500


Constant
Identifier
String-literal
17%
20%
1%
Keyword
3%
24%
Operador
35%
Punctuator