# Instituto Tecnológico de Costa Rica

## Compiladores e Intérpretes
## Proyecto #2: Analizador Léxico
## Profesor: Francisco Torres

Dennisse Rojas Casanova
Treicy Sánchez Gutiérrez

25 de Mayo, 2016

# Análisis Léxico y Flex

El Análisis Léxico consiste en descomponer un fuente de entrada en categorías léxicas mínimas llamadas tokens.Un programa en Flex consiste básicamente en una lista de expresiones regulares que definen acciones a ejecutar cuando ocurre un match.

```
void   open_ file( ) {
       char   filename  [ 100 ] = ;
       printf (   Enter  a  value  :   ) ;
       scanf( % s, filename) ;
       file  =   fopen (   filename,   r  ) ;
}
void   read_ file( ) {
  open_ file( ) ;
```

```
  if   ( file) {
    len  =   ftell( file) ;
    / / token_ buffer=   malloc ( sizeof ( char )   * ( len+ 1 ) ) ;
  }
  else {
    printf ( Problema  al  abrir  el  archivo\ n) ;
    exit ( 0 ) ;
  }
}
```

```
void close_file() {
  fclose (file);
}
void buffer_char (char c) {
  token_buffer[charPos++] = c;
}
```

```
void clear_token_buffer( ) {
  memset ( token_buffer, 0 , 5 ) ;
  charPos = 0 ;
}
token check_reserved( ) {
  int letter, c;
  bool reserved;
  / / Recorrer el token buffer, revisando su primera letra for
( letter= 0 ; letter < len_tb; letter++ ) {
```

```
    if  (   'B' ==   toupper ( token_ buffer[ letter] ) ) {
      reserved  =   true ;
      for ( c  =   0 ;  c  <   5 ;  c++ ) {
        if  ( begin_ buffer[ c]  !=   toupper ( token_ buffer[
letter++ ] ) ) {
          reserved  =   false ;
          break ;
        }
      }
      if ( reserved  ==   true ) {
          BEGIN;
```

```
    } else {
        ID;
    }
    break ;
  } else if ( 'E' == toupper ( token_buffer[ letter] ) ) {
    reserved = true ;
    for ( c = 0 ; c < 3 ; c++ ) {
      if ( end_buffer[ c] != toupper ( token_buffer[ letter++
] ) ) {
        reserved = false ;
```

```
        break ;
      }
    }
    if ( reserved  ==   true ) {
        END;
    }  else {
        ID;
    }
    break ;
} else  if  ( ' W'  ==  toupper ( token_ buffer[ letter] ) ) {
```

```
        reserved = true;
        for (c = 0; c < 5; c++) {
          if (write_buffer[c] != toupper(token_buffer[
letter++])) {
            reserved = false;
            break;
          }
        }
        if (reserved == true) {
            WRITE;
        } else {
```

```
            ID;
        }
        break ;
    } else if ( 'R' == toupper ( token_buffer[ letter] ) ) {
        reserved = true ;
        for ( c = 0 ; c < 4 ; c++ ) {
            if ( read_buffer[ c] != toupper ( token_buffer[
letter++ ] ) ) {
                reserved = false ;
                break ;
            }
```

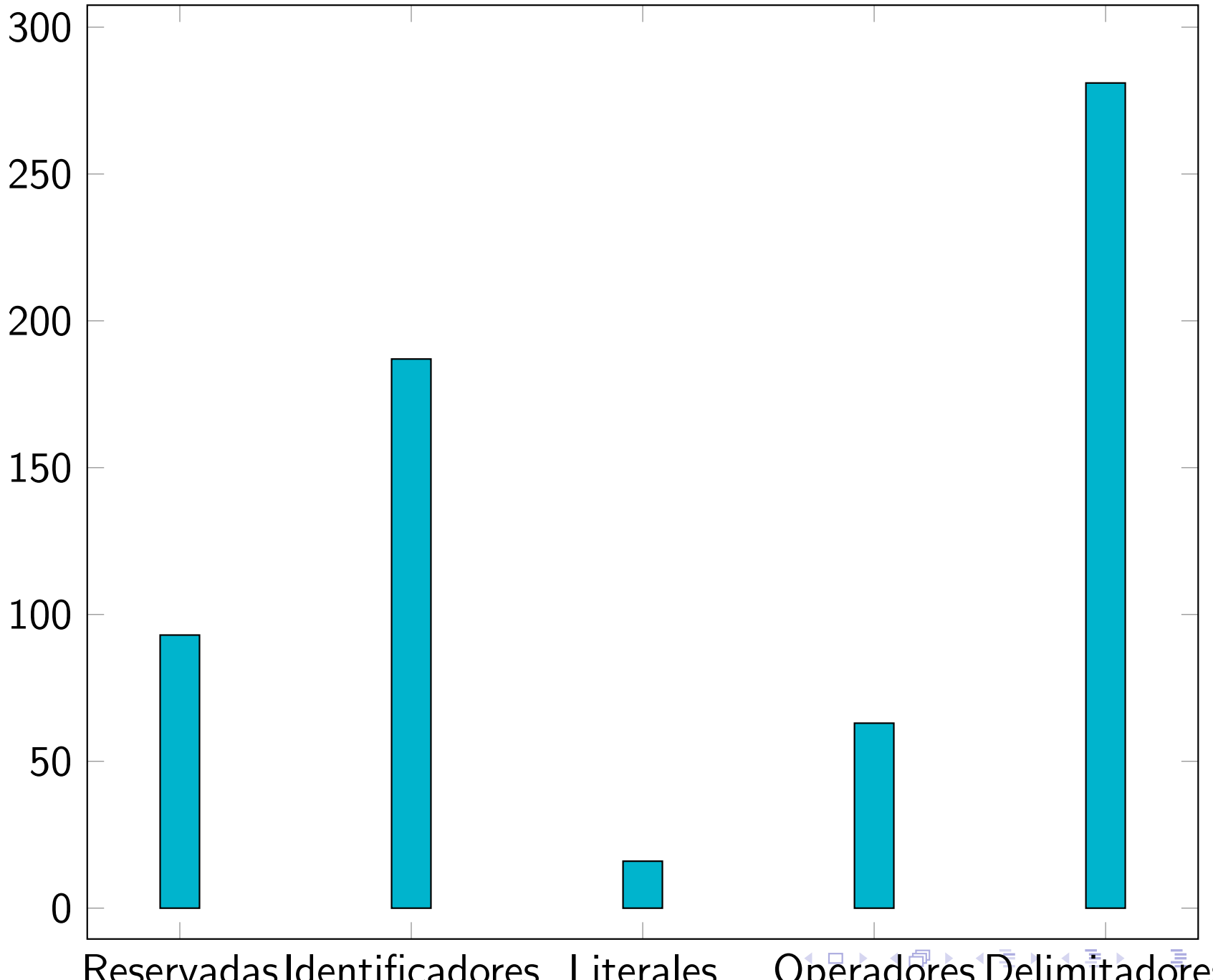
```
    }
    if ( reserved  ==   true ) {
        READ;
    }  else {
        ID;
    }
    break ;
}  else {
    ID;
}
```

```
  }
}
void lexical_error(int character) {
  printf(LEXICAL ERROR %d\n, character);
}
void print_token_buffer() {
  int i;
  printf(IMPRIMIENDO TOKEN BUFFER\n);
  //Prueba para ver que hay en el token_buffer.
```

```
  for ( i = 0 ; i <= len_ token_ buffer; i++ ) {
    printf ( % d\ n, token_ buffer[ i] ) ;
  }
}
void get_ tokens( ) {
  read_ file( ) ;
  token ejemplo;
  while ( filePos != len) {
    ejemplo = scanner( ) ;
```

```
    printf ( token   % d\ n,   ejemplo  ) ;
  }
}
```

# Histograma


300
250
200
150
100
50
0
Reservadas
Identificadores
Literales
Operadores
Delimitadores