Compiladores e Intérpretes
Informe de la Segunda Entrega

Diego Marcovecchio (LU: 83815) Leonardo Molas (LU: 82498)

2 de Septiembre de 2010

Índice general

Introducción

Descripción

Esta entrega consiste de un analizador léxico para un programa de Mini-Pascal. Cada *lexema* reconocido en el programa fuente es analizado, transformado al tipo de *token* que corresponda, e impreso a la salida especificada junto con su número de línea.

El programa tiene un nivel moderado de reconocimento de errores, permitiendo la detección de errores como un comentario abierto al finalizar el archivo.

El analizador léxico fue desarrollado utilizando únicamente **Python 2.7**[1] y algunas de sus librerías asociadas (**re**[2] y una modificación propia de **shlex**[3]).

[1] **Python** es un lenguaje de programación interpretado y multiplataforma. Para más información, dirigirse a la página oficial: `http://www.python.org/`

[2] **re** es una librería de Python que permite el reconocimiento de expresiones regulares. Su documentación puede ser vista en: `http://docs.python.org/library/re.html`

[3] **shlex** es una librería de Python para procesar comandos de consola. Nos basamos en su código fuente y realizamos algunas mejoras para procesar el stream de caracteres de entrada. La documentación de la versión original puede ser encontrada en: `http://docs.python.org/library/shlex.html`

Capítulo 1

Modo de uso

1.1. Requerimientos

La versión "`.exe`" del Analizador Léxico necesita una serie de librerías para funcionar, que fueron incluidas en la carpeta donde se encuentra el mismo ejecutable. Estas son:

- `python27.dll`
- `msvcr90.dll`
- `bz2.pyd`
- `select.pyd`
- `unicodedata.pyd`
- `library.zip` (que contiene las librerías de Python utilizadas)

1.2. Ejecución

lexan [−h|−−help] <IN_FILE> [<OUT_FILE>]

Argumentos:

<`IN_FILE`> El archivo de Pascal de entrada.

Argumentos opcionales:

<`OUT_FILE`> El archivo opcional de salida.

`-h, --help` Muestra la ayuda por pantalla.

Por ejemplo:

Lexan ejemplo1.pas output.txt

En este caso, el programa leerá el archivo `ejemplo1.pas` y devolverá el resultado en `output.txt`.

1.3. Formato de salida

El Analizador Léxico devuevle la información en una tabla, donde sus columnas indican el lexema (`LEXEME`), token (`TOKEN`) y número de línea (`LINE NUMBER`), como se puede ver en el siguiente ejemplo:

```
Starting file lexical analysis...

-------------------------------------------------------------------------------------------
|               LEXEME              |               TOKEN               |    LINE NUMBER     |
-------------------------------------------------------------------------------------------
|              program              |             <PROGRAM>             |         1          |
|              ejemplo1             |            <IDENTIFIER>           |         1          |
|                 ;                 |            <SEMI_COLON>           |         1          |
|               const               |              <CONST>              |         2          |
|                 Z                 |            <IDENTIFIER>           |         2          |
|                 =                 |              <EQUAL>              |         2          |
|                 4                 |              <NUMBER>             |         2          |
|                 ;                 |            <SEMI_COLON>           |         2          |

                                                    ...

|                 c                 |            <IDENTIFIER>           |         14         |
|                 :=                |            <ASSIGNMENT>           |         14         |
|                'd'                |               <CHAR>              |         14         |
|                 ;                 |            <SEMI_COLON>           |         14         |
|                end                |               <END>               |         16         |
|                 .                 |           <END_PROGRAM>           |         17         |
|                                   |               <EOF>               |         17         |
-------------------------------------------------------------------------------------------

Finished lexical analysis succesfully!
```

Capítulo 2

Lenguaje

2.1. Alfabeto de entrada

```
<letter>::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
    Q | R | S | T | U | V | W | X | Y | Z | a | b | c | d | e | f | g | h |
    i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<digit>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<special symbol> ::=  + | - | * | = | <> | < | > | <= | >= | ( | ) | [ | ]
   | { | } | := | . | , | ; | : | div | or | and | not | if | then | else |
    while | do | begin | end | const | var | type | array | function |
   procedure | program
```

2.2. Errores detectados

El analizador léxico tiene un nivel moderado de detección de errores. Entre éstos, se encuentran:

- Caracter no reconocido: si se intenta ingresar un caracter que no pertenece al alfabeto, como "@", se producirá un error.
- Comentarios abiertos: si el programa fuente tiene un comentario sin cerrar cuando termina el archivo, se informa el error.
- Números mal formados: si se intenta ingresar un número como 38a7, se informará el error.
- Si el archivo fuente especificado no existe, se informará el error.

Los mensajes de error mostrarán el número de línea, con el lexema que generó el error, cuando corresponda.

Capítulo 3

Analizador léxico

3.1. Palabras reservadas

El Analizador Léxico reconoce las siguientes palabras reservadas especificadas en la tabla 3.1

Palabras reservadas
program
type
const
var
array
of
function
procedure
begin
end
while
do
if
then
else
div
not
or
and
true
false

Cuadro 3.1: Palabras reservadas

3.2. Tokens

El Analizador Léxico reconoce los tokens especificados en la tabla 3.2.

Token	**Expresión Regular**
Identifier	`[a-zA-Z][a-zA-Z0-9]*`
Number	`[0-9]+`
Char	`'[a-zA-Z0-9]'`
RelOp	`<|>|<=|>=`
Arith_Op	`+|-|*`
Un_LogOp	`not`
Bin_LogOp	`or|and`
Equal	`=`
Type_Declaration	`:`
Assignment	`:=`
Comma	`,`
Semicolon	`;`
End_Program	`.`
Subrange_Separator	`..`
EOF	
Open_Parenthesis	`(`
Close_Parenthesis	`)`
Open_Bracket	`[`
Close_Bracket	`]`
Program	`program`
Type	`type`
Const	`const`
Var	`var`
Function	`function`
Procedure	`procedure`
Array	`array`
Of	`of`
Begin	`begin`
End	`end`
While	`while`
Do	`do`
If	`if`
Then	`then`
Else	`else`

Cuadro 3.2: Tokens