Universidad de La Laguna

E. T. S. DE INGENIERÍA INFORMÁTICA
Departamento de E.I.O. y Computación
Compiladores

PRÁCTICA 1: Diseño e implementación de un analizador léxico para *Pascal-*

Factor de ponderación [0-10]: 8

1.2. Objetivo

La práctica consiste en escribir en Pascal un analizador léxico que reconozca los tokens del lenguaje *Pascal-* (este lenguaje es un subconjunto de Pascal). En la práctica se pretende determinar los tokens del lenguaje y manipular tablas de símbolos.

1.3. Requisito de codificación

Cada fichero de código fuente de los que se utilicen en la práctica ha de llevar comentarios de cabecera. También es imprescindible que cada función lleve comentarios de cabecera con una breve indicación de la operación que implementa.

1.4. Descripción

El analizador léxico ha de reconocer las palabras reservadas que aparecen en la Tabla 1.1 (tanto en mayúsculas como en minúsculas o mezcla de ambas).

También ha de reconocer los símbolos que aparecen en la Tabla 1.2 Ha de reconocer números enteros e identificadores (del tipo de los de Pascal). Si se introduce un símbolo que no se corresponde con ninguno de los anteriores se ha de devolver el token TOKEN_ERROR.

El Listado 1.1 contiene la lista de todos los componentes léxicos de *Pascal-*.

and	**array**	**begin**	**const**	**div**
do	**else**	**end**	**if**	**mod**
not	**of**	**or**	**procedure**	**program**
record	**then**	**type**	**var**	**while**

Tabla 1.1: Palabras reservadas

+	-	*	<	=
>	≤	<>	≤	:=
(	)	[	]	,
.	:	;	..	

Tabla 1.2: Signos especiales

```
AND, ARRAY, ASTERISK, BECOMES, BEGIN,
COLON, COMMA, CONST, DIV, DO, DOUBLEDOT,
ELSE, END, ENDTEXT, EQUAL, GREATER,
ID, IF, LEFTBRACKET, LEFTPARENTHESIS,
LESS, MINUS, MOD, NOT, NOTEQUAL,
NOTGREATER, NOTLESS, NUMERAL, OF, OR,
PERIOD, PLUS, PROCEDURE, PROGRAM,
RECORD, RIGHTBRACKET, RIGHTPARENTHESIS,
SEMICOLON, THEN, TYPE, TOKEN_ERROR, VAR,
WHILE
```

Listado 1.1: Tokens

El analizador léxico ignorará los comentarios, que estarán delimitados por los símbolos "{" y "}" y también "(*" y "*)".

También llevará la cuenta del número de línea y número de columna (posición dentro de la línea) del programa fuente que está siendo analizado. Esta información será útil a la hora de generar posibles mensajes de error.

Los tokens `ID` y `NUMERAL` tendrán un atributo asociado. Utilizaremos el tipo de datos del Listado 1.2 para representar los atributos.

```
typedef union {
  WordPointer ptr;  /* Puntero a la TS */
  int value;
} argtype;
```

Listado 1.2: Atributos de los tokens

Los números en formato decimal devolverán su valor en un atributo `argvalue.value` (que será de la clase `argtype`).

Se considerará un identificador toda aquella palabra que comience con una letra y vaya seguida de letras, dígitos o caracteres "underscore" (guión bajo). La cadena asociada con dicho identificador se almacenará en la tabla de símbolos. A cada identificador se le asociará como atributo un puntero que indica la posición correspondiente en la tabla de símbolos. En `argvalue.ptr` se devolverá el atributo asociado a cada identificador mientras que en `argvalue.value` se devolverá el atributo asociado a un token `NUMERAL`.

Para representar la tabla de símbolos se utilizará la estructura de datos del Listado 1.3.

```
typedef struct WordRecord {
  struct WordRecord *NextWord;
  char *lexeme;
  bool IsName;
  int Index;
} WordRecord;
typedef WordRecord* WordPointer;
WordPointer ts[MAX_KEY];     /* Tabla de simbolos*/
```

Listado 1.3: Estructura de la Tabla de Símbolos

Se utilizará la función hash del Listado 1.4 (o cualquier otra que tenga un buen comportamiento).

```
int hash(char *WordText) {
#define W (MAXINT-127) /* Evita desbordamientos */
  int sum = 0, i;

  for (i = 0; i < strlen(WordText); i++)
    sum = (sum + (int)WordText[i]) %W;
  return (Sum %MAX_KEY);
}
```

Listado 1.4: La función Hash

La tabla de símbolos estará inicializada a vacío y lo primero que se inserta en ella son las palabras reservadas y los identificadores estándar (esta tarea la realizará la función `Initialize()`). Los identificadores estándar y su número (index) asociado aparecen en la Tabla 1.3.

El resto de identificadores que aparezcan en el programa fuente tendrán valores de `index` diferentes para cada identificador, comenzando en 7.

integer	1
boolean	2
false	3
true	4
read	5
write	6

Tabla 1.3: Identificadores estándar

1. Escribir las funciones para el tratamiento de la tabla de símbolos.

   a) Una rutina que busque una cadena de caracteres `WordText` en la tabla de símbolos:

   ```
   void Search(char *WordText, bool *IsName,
      WordPointer *Ptr);
   ```

   Se recomienda implementar esta función de la siguiente forma:

   - Calcular el valor de la función hash asociado a la cadena `WordText`.
   - Buscar `WordText` en la lista correspondiente de la tabla de símbolos.
   - Si la cadena `WordText` está en la tabla de símbolos, se devolverá un puntero a la estructura en la que está almacenada la información que tiene asociada (`Ptr`) y un valor booleano que indique si se trata de una palabra reservada o de un identificador (`IsName`).
   - Si no está en la tabla, se inserta en ella como si fuera un identificador y se devuelven los valores como en el caso anterior.

   b) Una función que inserte en la tabla de símbolos una cadena de caracteres `WordText`. La cadena `WordText` puede corresponder a un identificador o a una palabra reservada. En el caso de un identificador se almacenará en el campo `Index` de la estructura de datos, un entero que indique el ordinal que le corresponde dentro del programa fuente, mientras que a cada palabra reservada se le asociará el ordinal que le corresponde en la definición del tipo enumerado token. Por ejemplo: a la palabra reservada `AND` le corresponde el índice 0, a `ARRAY` le corresponde 1, a `BEGIN` el 4 y así sucesivamente. En cuanto a los identificadores,

al identificador estándar `integer` le corresponde el 1, al identificador `boolean` el 2, y así sucesivamente según el orden en que vayan apareciendo nuevos identificadores en el programa fuente. Habrá que utilizar un atributo para llevar la cuenta del número de identificadores que van apareciendo en el programa fuente.

```
void Insert(bool IsName, char *WordText, int
   Index, int KeyNo, WordPointer *Ptr);
```

Para construir esta función, se puede proceder del siguiente modo: Insertar la estructura de datos que representa a la palabra en la lista correspondiente de la tabla de símbolos. En dicha estructura de datos se almacenan los valores de los parámetros que indican si se trata de una palabra reservada o identificador (`IsName`), el índice que se le asigna a cada identificador o palabra reservada (`Index`) y el lexema correspondiente (`lexeme`).

El parámetro `KeyNo` indica la clave hash correspondiente a la lista en la que se tiene que insertar la cadena.

En el parámetro ptr se devuelve el puntero a la estructura de datos que almacena la información relacionada con `WordText`.

2. Escribir una función que ignore los comentarios encerrados entre los símbolos "{" y "}" y también entre los símbolos "(*" y "*)". Se han de permitir comentarios anidados, esto es, comentarios de la forma:

   ```
   { este comentario tiene { este otro comentario }
      dentro }
   ```

   La cabecera de esta función será:

   ```
   void Comment(void);
   ```

   Se ha de emitir un mensaje de error cuando no se encuentre el final de un comentario.

3. Escribir un método que realice el análisis léxico del fichero de entrada:

   ```
   token yylex(argtype *argvalue);
   ```

   Para escribirla, se puede utilizar el siguiente esquema:

- Ignorar los separadores (espacios en blanco, tabuladores y retornos de carro) así como los comentarios. Si se cambia de línea aumentar el contador de líneas (atributo `line_num`).
- Reconocer los números. Devolver como atributo asociado al token `NUMERAL` el valor del número leído. Emitir un mensaje de error si el valor del número leído desborda a un entero.
- Reconocer las cadenas de caracteres. Si se trata de una palabra reservada, devolver el token correspondiente. En el caso de un identificador, devolver el token `ID` y como atributo el puntero a la entrada de la tabla de símbolos que lo contiene.
- Reconocer cada uno de los signos de puntuación del lenguaje.

Téngase en cuenta que:

El carácter del fichero de entrada que se analiza en cada momento estará almacenado en el atributo .

El analizador léxico, al inicializarse calculará el tamaño en bytes del fichero fuente, alojará memoria suficiente para almacenarlo en memoria y lo leerá. `nextchar()` se encargará de devolver un nuevo carácter, almacenarlo en el atributo `ch` y avanzar a la siguiente posición del fichero de entrada.

Para el reconocimiento de algunas clases de caracteres, se pueden utilizar conjuntos de caracteres, que se han de implementar (se puede usar la implementación que se hizo en la asignatura TALF).

Utilizar programas *Pascal-* implementados a propósito para comprobar el analizador léxico diseñado. Aparte de comprobar el analizador en "condiciones normales", ponerlo también a prueba con programas erróneos, para comprobar un funcionamiento adecuado del analizador léxico.