Atizapán de Zaragoza, a 21 de enero de 2016

Tecnológico de Monterrey
Campus Estado de México
Escuela de Diseño, Ingeniería y Arquitectura
Departamento de Tecnologías de la Información y Computación

Diseño de compiladores
Definición del lenguaje de programación `C--`
Primera parte
Léxico y sintaxis

Profesor: Edgar Emmanuel Vallejo Clemente

1 Introducción

En este curso desarrollarás un compilador para un lenguaje de programación imperativa simple llamado `C--`. Este documento presenta una especificación para el léxico y la sintaxis del lenguaje, la cual será requerida para el desarrollo del primer avance del proyecto final.

2 Consideraciones del léxico

Las siguientes son palabras clave; todas ellas son palabras reservadas:

```
void int double bool string while for if else return print readint readline
```

Un identificador es una secuencia de letras, dígitos, y guiones bajos, que comienzan con una letra. `C--` es sensible a las mayúsculas y minúsculas. (Ejemplo `while` es una palabra clave, pero `WHILE` es un identificador). `Foo` y `foo` son dos identificadores distintos. Los identificadores pueden tener a lo más 31 caracteres de longitud.

Los espacios en blanco (espacios, tabuladores, y cambios de línea) se utilizan para separar los *tokens*, pero en otro caso, se ignoran. Las palabras clave y los identificadores deben separarse por medio de espacios en blanco o por un *token* que no es un identificador o una palabra clave. `ifintwhile` es un identificador, no tres palabras clave, pero `if(23bool` se escanéa como cuatro tokens.

Una constante booleana es `true` o `false`. Al igual que las palabras clave, estas palabras son reservadas .

Una constante entera puede especificarse en decimal (base 10) o hexadecimal (base 16). Un entero decimal es una secuencia de dígitos decimales `0-9`. Un entero hexadecimal debe comenzar con `0X` o `0x` seguido de una secuencia de dígitos hexadecimales. Los dígitos hexadecimales incluyen a los dígitos decimales y a las letras mayúsculas `A-F` o minúsculas `a-f`. Por ejemplo, los siguientes son enteros válidos:

```
8, 012, 0x0 0X12aE
```

Una constante doble es una secuencia de dígitos decimales (al menos uno), un punto decimal, seguido de cualquier secuencia de dígitos (al menos uno). (Ejemplo: `.75` y `75.` no son válidos, pero `0.75` sí es válido). Un doble también puede tener un exponente opcional (Ejemplo `75.5E+2`). Para un doble con exponente, se requiere el punto decimal, el signo del exponente es opcional (si no se especifica, se asume `+`), la `E`, que puede ser mayúscula o minúscula, y una secuencia de dígitos decimales (al menos uno). Como antes, `.75E+2` y `75.E+2` son inválidos, pero `75.0E+2` es válido. Se permiten ceros a la izquierda en la mantisa y el exponente.

Una constante string es una secuencia de caracteres delimitada por comillas dobles. La secuencia puede incluir cualquier caracter excepto el cambio de línea o la comilla doble. Un string debe comenzar y finalizar en la misma línea; no se puede segmentar en varias líneas:

```
"Esta no es una
constante string válida"
```

Los operadores y delimitadores utilizados en el lenguaje son:

```
+ - * / % < <= > >= = == != && || !  ; , [ ] ( ) { }
```

Los comentarios de una línea comienza con `//` y se extiende hasta el fin de la línea. Los comentarios estilo lenguaje C comienzan con `/*` y terminan con el primer `*/` subsecuente. Se permite cualquier símbolo dentro de un comentario, excepto la secuencia de terminación de comentario `*/`.

3 Gramática de referencia

La gramática de referencia se especifica utilizando una notación similar a la BNF. La notación utilizada es:

`x`	(`typewriter`) significa que `x` es un terminal (es decir, un *token*). Los terminales se escriben en minúsculas
X	(*itálica*) significa que X es una categoría sintáctica (una variable de la gramática). Todas las categorías sintácticas comienzan con mayúsculas.
x	(*itálica*) significa que x es cualquier símbolo de la gramática (variable o terminal).
$<x>$	significa cero o una ocurrencia de x
x^*	significa cero o más ocurrencias de x
x^+	significa una o más ocurrencias de x
$x^+,$	significa una lista de una o más ocurrencias de x separadas por comas (entre las x's)
|	separador de las diferentes reglas alternativas de X
ε	string vacío

Para mejorar la legibilidad, se presentan los operadores por medio del lexema que lo denota, en vez de la categoría de *token* que regresa el analizador de léxico:

Programa	::=	*Decl*$^+$
Decl	::=	*DeclVariable* | *DeclFuncion*
DeclVariable	::=	*Variable* `;`
Variable	::=	*Tipo* `identificador`
Tipo	::=	`int` | `double` | `bool` | `string [ constanteentera ]`
		| *Tipo* `[ constanteentera ]`
DeclFuncion	::=	*Tipo* `identificador (`*Formales*`)` *BloqueInstr*
		| *void* `identificador (`*Formales*`)` *BloqueInstr*
Formales	::=	*Variable*$^+$`,` | ε
BloqueInstr	::=	{ *DeclVariable** *Instr**}
Instr	::=	<*Expr*> `;` | *InstrIf* | *InstrWhile* | *InstrFor*
		| *InstrReturn* | *InstrPrint* | *BloqueInstr*
InstrIf	::=	`if(`*Expr*`)` *Instr* <`else` *Instr*>
InstrWhile	::=	`while(`*Expr*`)` *Instr*
InstrFor	::=	`for(`<*Expr*>`;` *Expr*`;` <*Expr*>`)` *Instr*
InstrReturn	::=	`return` <*Expr* >`;`
InstrPrint	::=	**print**(*Expr*$^+$`,`)`;`
Expr	::=	*ValorL* `=` *Expr* | *Constante* | *ValorL* | *Llamada* | (*Expr*)
		| *Expr* `+` *Expr* | *Expr* `-` *Expr* | *Expr* `*` *Expr* | *Expr* `/` *Expr*
		| *Expr* `%` *Expr* | `-` *Expr* | *Expr* < *Expr* | *Expr* <= *Expr*
		| *Expr* > *Expr* | *Expr* >= *Expr* | *Expr* `==` *Expr* | *Expr* `!=` *Expr*
		| *Expr* `&&` *Expr* | *Expr* || *Expr* | `!` *Expr* | `readint ( )`
		| `readline( )`
ValorL	::=	`identificador` | *Expr*`[` *Expr* `]`
Llamada	::=	`identificador(` *Reales* `)`
Reales	::=	*Expr*$^+$`,` | ε
Constante	::=	`constanteentera` | `constantedoble` | `constantebooleana`
		| `constantestring`