LENGUAJES DE PROGRAMACIÓN
Y
PROCESADORES DE LENGUAJES

Construcción de un compilador

`MenosC`

Parte-I: Análisis léxico sintáctico

El objetivo principal es la construcción de un compilador completo,

para un Lenguaje de Programación de alto nivel, sencillo pero no trivial

MenosC

`Parte I` Construcción del analizador léxico-sintáctico

$\Rightarrow$ límite de entrega **29 de octubre de 2023**

`Parte II` Construcción del analizador semántico

$\Rightarrow$ límite de entrega **10 de diciembre de 2023**

`Parte III` Construcción del generador de código intermedio

$\Rightarrow$ límite de entrega **23 de enero de 2024**

[para la recuperación: **9 de febrero de 2024**]

Especificación Léxica

- Los identificadores son cadenas de letras (incluyendo "_") y dígitos, que comienzan siempre por una letra. Debe distinguirse entre mayúsculas y minúsculas.
- Las palabras reservadas se deben escribir en minúscula.
- En un programa fuente puedan aparecer constantes enteras y reales; por ejemplo: `28` `28.` `.55` `28.55`
- El signo $+$ (ó $-$) de las constantes numéricas se tratará como un símbolo léxico independiente.
- Los espacios en blanco, retornos de linea y tabuladores deben ignorarse.
- Los comentarios deben ir precedidos por la doble barra (//) y terminar con el fin de la linea. Los comentarios no se pueden anidar.

Especificación Sintáctica

programa	→	listDecla
listDecla	→	decla | listDecla decla
decla	→	declaVar | declaFunc
declaVar	→	tipoSimp **id** ; | tipoSimp **id** [**cte**] ; | **struct** { listCamp } **id** ;
tipoSimp	→	**int** | **bool**
listCamp	→	tipoSimp **id** ; | listCamp tipoSimp **id** ;
declaFunc	→	tipoSimp **id** (paramForm) { declaVarLocal listInst **return** expre ; }
paramForm	→	ϵ | listParamForm
listParamForm	→	tipoSimp **id** | tipoSimp **id** , listParamForm
declaVarLocal	→	ϵ | declaVarLocal declaVar
listInst	→	ϵ | listInst inst
inst	→	{ listInst } | instExpre | instEntSal | instSelec | instIter

Especificación Sintáctica

instExpre → expre ; | ;

instEntSal → **read** (**id**) ; | **print** (expre) ;

instSelec → **if** (expre) inst **else** inst

instIter → **while** (expre) inst

expre → expreLogic | **id** = expre | **id** [expre] = expre | **id** . **id** = expre

expreLogic → expreIgual | expreLogic opLogic expreIgual

expreIgual → expreRel | expreIgual opIgual expreRel

expreRel → expreAd | expreRel opRel expreAd

expreAd → expreMul | expreAd opAd expreMul

expreMul → expreUna | expreMul opMul expreUna

expreUna → expreSufi | opUna expreUna | opIncre **id**

Especificación Sintáctica

expreSufi	→	const | (expre) | **id** | **id** opIncre | **id** . **id**
		| **id** [expre] | **id** (paramAct)
const	→	**cte** | **true** | **false**
paramAct	→	ϵ | listParamAct
listParamAct	→	expre | expre , listParamAct
opLogic	→	&& | ||
opIgual	→	== | !=
opRel	→	> | < | >= | <=
opAd	→	+ | -
opMul	→	* | /
opUna	→	+ | - | !
opIncre	→	++ | --

Recomendaciones para la codificación

Haced que vuestro código sea más fácil de leer por otros es siempre una buena idea, y adoptar un buen estilo de codificación os ayudará a lograrlo.

Unas recomendaciones básicas podrían ser:

- Usar sangrías de pocos espacios, no tabuladores.
- Recortar las líneas para que no superen los 80 caracteres.
- Cuando sea posible, poner comentarios en una sola línea.
- No usar codificaciones extravagantes, recordad que el proyecto es en equipo y el código debe ser compartido.

Recomendaciones para la codificación

Para terminar recordad la lista de los famosos *pecados capitales*, tomados del libro Writing Interactive Compilers and Interpreters, Wiley & Sons, 1979. de P.J.Brown.

1. Codificar antes de pensar.
2. Asumir que el usuario tiene todo el conocimiento que tiene el escritor del compilador.
3. No escribir la documentación adecuada.
4. Ignorar los estándares del idioma.
5. Tratar el diagnóstico de errores como una ocurrencia tardía.
6. Equiparar lo improbable con lo imposible.
7. Hacer que la codificación del compilador dependa del formato de los datos.
8. Fingir que atiende a todos los problemas al mismo tiempo.
9. Valorar la belleza de la codificación por encima de la usabilidad del compilador.
10. Dejar que cualquier error pase desapercibido.
11. Dejar que los usuarios encuentren los errores en su compilador.