Convenciones de lenguaje L2y3

El lenguaje L2y3 está siendo desarrollado como proyecto un proyecto académico de dos estudiantes de la universidad Simón Bolívar, es un pequeño lenguaje imperativo orientado a la fácil lectura y comprensión del código, en este documento se encuentran las diferentes convenciones sobre el diseño de este lenguaje:

**Convenciones sobre el lenguaje:**

* + Todos los archivos de código del lenguaje deben tener la extensión .ly
  + Todos los archivos del lenguaje solo pueden contener lo siguiente:
    - Instrucciones del tipo importación de código.
    - Declaración de variables globales, tipos compuestos o subrutinas.

Solamente se pueden realizar esas instrucciones en la raíz del archivo.

* Al comienzo de la ejecución de cualquier programa de este lenguaje se ejecutara la subrutina de nombre ‘Main’, en caso de no existir el programa terminara de inmediato.
* Si en este documento aparece que el nombre o algún otro valor de algo es igual a alguna expresión regular, se asume que quiere decir la expresión regular menos cualquier palabra reservada reconocida por la misma.
* Todas la palabras reservadas del sistema comienzan por una letra mayúscula, en caso de que en el documento aparezca una palabra reservada que no cumple esta condición, es un error del documento.

**Tipos Básicos:**

Los tipos básicos son:

**Enteros (Int):** Representación de los números enteros

Tamaño: 32 bits (1 bit para el signo, el resto para el numero)

Operaciones definidas: Suma ‘+’, Resta ‘-‘, Multiplicación ‘\*’, División entera ‘/’, Menor que ‘ <’, Mayor que ‘ >’, Menor o igual que ‘<=’, Mayor o igual que ‘>=’, Resto ‘%’, Igualdad ‘ ==’, Diferencia ‘!=’.

**Booleanos (Bool):** Representación de una variable booleana (True o False)

Tamaño: 1 byte

Operaciones definidas: ‘Y’ lógico con cortocircuito ‘&&’, ‘O’ lógico con cortocircuito ‘||’, Igualdad ‘==’, Diferencia ‘!=’, Negación ‘!’.

**Carácter (Char):** Representación de un carácter ASCII

Tamaño: 1 byte

Operaciones definidas: ninguna.

**Reales (Real):** representación de un número real

Tamaño: 32 bits

Operaciones definidas: Suma ‘+’, Resta ‘-‘, Multiplicación ‘\*’, División entera ‘/’, Menor que ‘ <’, Mayor que ‘ >’, Menor o igual que ‘<=’, Mayor o igual que ‘>=’, Igualdad ‘ ==’, Diferencia ‘!=’.

**Convenciones sobre los tipos básicos:**

Todos los tipos básicos de L2y3 se manejan utilizando el esquema de valor.

Se puede convertir los siguientes tipos mediante un casteo explicito:

Int -> Char y viceversa

Real -> Int y viceversa

**Variables:**

Las variables se declaran de la siguiente forma:

<tipo> <nombre\_1> … <nombre\_n> := <valor\_1> … <valor\_n>;

**Reglas sobre variables**

* + Los nombres de las variables solo pueden ser del tipo: [$@]?[a-z][\_a-zA-Z]\*
  + Si el nombre de una variable comienza por ‘$’, es una constante.
  + Si comienza por ‘@’ es estática.
  + Si comienza de alguna otra manera es una variable normal
  + Cualquier variable solo es válida una vez que llegue a la línea de código que contiene su declaración.
  + Las variables pueden ser de cualquier tipo básico o compuesto.
  + Ninguna variable puede ser utilizada a menos que se le haya inicializado primero.
  + Las variables no inicializadas deben ser detectadas a tiempo de compilación y se debe abortar enviando un error.

**Tipos Compuestos:**

**Convenciones sobre los tipos compuestos:**

* + Todos los tipos compuestos se manejan mediante el esquema de referencia.
  + No existen operaciones predefinidas para los tipos complejos, cualquier operación necesaria se debe hacer mediante subrutinas.
  + Los tipos compuestos en L2y3 son los siguientes:

**Arreglos:**

* + Pueden ser de cualquier tipo básico o compuesto.
  + Se declaran de la forma <tipo>[<tamaño>] <nombre1> ;
  + El tamaño del arreglo puede ser definido estáticamente o dinámicamente, y la memoria del arreglo será reservada cuando sea conveniente en ambos casos.
  + En caso de que se trate de accesar un índice que no se encuentre mediante el rango del arreglo, se debe abortar con un error.
  + Las variables internas dentro de un arreglo cumplen todas las reglas sobre inicialización antes descritas en la sección de variables.

**Registros:**

* Son de la forma:

Typedef <nombre> {

<Declaración de variable 1>;

…

<Declaración de variable n>;

}

* + Pueden contener variables de cualquier tipo y estas pueden ser inicializadas dentro de su declaración o fuera de este sin ningún problema.
  + Los nombres de los registros son del tipo [A-Z][\_a-zA-Z]\*
  + Los nombres de las variables dentro de un registro son independientes de los nombres de las variables externas a él y siguen las convenciones de los nombres de las variables normales.
  + Para tener acceso a los atributos de un registro se utiliza el operador ‘.’ Sobre la variable del tipo del registro.

**Variantes:**

* Son de la forma:

Union <nombre> {

<tipo\_1> <Palabra\_clave\_1>;

…

<tipo\_n> <Palabra\_clave\_n>;

}

* Las nombres y palabras clave son de la forma: [A-Z][\_a-zA-Z]\*
* Los variantes son del tamaño de su tipo más grande.
* Para acceder a la información de un variante se debe utilizar el operador ‘.’ Para indicar que tipo se va utilizar mediante la palabra clave, esto se aplica tanto como cuando se escribe como cuando se lee.
* Una vez que un variante contiene un tipo y se le trata de escribir algo de otro tipo sobre este, la información original se pierde sin aviso alguno.

**Strings:**

* + Son cadenas de caracteres.
  + Se definen de la forma: String <nombre> := “String entre comillas”; si se desea un string de tamaño dinámico, o Char[<Tamaño>] <nombre> := “String de longitud menor al tamaño”; si se desea un string de tamaño fijo.
  + Se puede accesar a cada carácter individual mediante <nombre>[<índice>] independientemente de cómo se halla declarado el string.

**Estruturas de control:**

**Convenciones sobre estructuras de control:**

* En todas las estructuras de control, tanto los paréntesis y las llaves que aparezcan indicados son obligatorios.
* Es posible tener variables locales normales en los bloques dentro de las estructuras de control, solamente cuando sea indicado explícitamente estas variables seguirán siendo accesibles una vez que termine el bloque de instrucciones.
* Las variables estáticas y constantes declaradas en un bloque de instrucciones, siguen siendo accesibles posteriormente a la ejecución del bloque.

**Condicionales:**

* Son de la forma:

If (<Condicion>) {

<Bloque de instrucciones.>

}Elseif(<Condicion>){

<Bloque de instrucciones.>

}Else{

<Bloque de instrucciones.>

}

* Solo se puede tener uno o ningún Else por cada If, en cambio se pueden tener ningún o infinitos Elseif por cada If.
* Las variables inicializadas en un If, que fueron declaradas fuera de ese If, solo se consideran inicializadas dentro de ese If, a menos que se inicialicen en todos los casos posibles, (Esta comprobación se realiza estaticamente).
* Las variables normales que se declaren en cualquier bloque de instrucción del If solo existen dentro de ese bloque independientemente de si una variable con el mismo nombre fue declarada en todos los casos posibles.
* En caso de que existan varias condiciones que se cumplan en un If, solo se ejecutara la primera y se dará por terminada la ejecución del mismo.

**Iteraciones:**

* Pueden ser de dos tipos While o For:
* Los While tienen la forma:

While ( <condición>){

<Bloque de instrucciones>

}

* Los For tienen la forma:

For(<variable>, <condición>,<instrucción>){

<Bloque de instrucciones>

}

* En ambos casos cualquier variable normal declarada dentro del bloque de instrucciones, existe una vez por cada iteración del bloque, es decir su valor se reinicia en cada iteración.
* En el For la variable utilizada puede ser declara dentro del paréntesis o antes de la declaración del For, en caso de que se declare en los paréntesis esta variable se considera local al For y en ambos casos es posible la escritura y lectura sobre la variable durante la iteración.

**Subrrutinas:**

**Convenciones sobre subrutinas:**

* Las subrutinas se declaran de la siguiente manera:

Function <tipo> <nombre>(<tipo1><parametro1>,…,<Tipo\_n><Parametro\_n>){

<Bloque de instrucciones>

}

* El tipo de la subrutina se refiere al valor del parámetro que va a regresar, este puede ser cualquier tipo ya sea básico o compuesto, también se permite colocar la palabra ‘Void’ para indicar que la subrutina no devuelve ningún parámetro.
* El nombre de la subrutinas es de la forma: [A-Z][\_a-zA-Z]\*
* Las variables locales declaradas dentro de una subrutina solo existen dentro de esta.
* Las variables estáticas o las constantes declaradas dentro de una subrutina, siguen existiendo una vez terminada su ejecución.
* Se utiliza la palabra reservada ‘Return’ para indicar el parámetro a regresar en caso de que exista, este debe ser del tipo indicado en la declaración de la función, en caso de no serlo se devuelve un error de tipos.
* Una vez se ejecute la instrucción Return, la ejecución de la subrutina termina.