Especificacion del Lenguaje *BLA*

Fernández, K. & Gyomrey, A.

El presente documento muestra la especificación del lenguaje *BLA*. *BLA* es un lenguaje imperativo fuertemente tipado. Su sintáxis y operaciones tiene elementos inspirados en C y otros lenguajes tales como Haskell, Python y R.

Consideraciones Léxicas

Las siguientes palabras son reservadas del lenguaje:

> `none`, `int`, `float`, `boolean`, `char`, `string`, `fun`, `for`, `fill`, `if`, `elif`, `else`, `return`, `continue`, `break`, `map`, `foldr`, `foldr`, `const`, `alias`

Estas palabras corresponden con las funciones nativas de *BLA* explicadas más adelante.

Un idenficador es una secuencia de letras y dígitos, empezando por una letra. *BLA* es sensitivo a capitalización, es decir, el uso de de mayúsculas y minúsculas hace variar un identificador. Por ejemplo, `foo` y `Foo` son identificadores distintos.

En *BLA* es posible escribir comentarios en bloque y de línea basados en la sintáxis de Haskell. Un comentario de línea empieza con dos guiones consecutivos `--` y finaliza con un fin de línea. Los comentarios de bloque empiezan con `{-` y terminan con `-}`. Cualquier símbolo es permitido en un comentario excepuanto la secuencia `-}`, la cual tiene la función única de finalizar el comentario actual.

Las constantes booleanas son `true` y `false` correspondientes respectivamente a los valores de verdadero y falso.

Un entero puede ser especificado con o sin su base. Un entero con base es una secuencia de dígitos decimales (0–9) y caracteres (a-zA-Z), seguido de un guión bajo (_) y de la base especificada en formato decimal. Los carácteres son válidos siempre que la base lo requiera. La base es un valor entre 1 y 36, ambos inclusive.

En caso de no indicar la base, se considera que el entero está expresado en base decimal. Los siguientes son ejemplos válidos de enteros:

> 10_2, 124_7, 2A_16, 2br_20, 00000_1

Un número flotante es una secuencia de dígitos, un punto, seguido de cualquier secuencia de dígitos. La segunda secuencia puede estar vacía. Adicionalmente, se pueden escribir en notación científica. El signo del exponente es opcional, en caso de no especificarse se asume positivo y `E` puede estar expresado en mayúsculas o minúsculas.

Son ejemplo de flotantes válidos:

4.67, 0.42, 3., 37.0, 23.4E+2, 42.8e–6

Una constante `string` es una secuencia de caracteres delimitados por comillas dobles. Los strings pueden contener cualquier caracter. Una cadena de caracteres puede empezar y terminar en líneas diferentes. Dos cadenas de caracteres contínuas se concatenan automáticamente. Es decir, el siguiente es un `string` válido.

```
"Este es un string válido. Nótese que las comillas que lo
delimitan estan
en líneas diferentes." "Este string es continuación del anterior."
"Y este también."
```

Los operadores y caracteres de puntuación usados por el lenguaje son los siguientes:

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

Gramática de *BLA*

TODO: definir la gramática. Idea: ir definiéndola a medida que vayamos estableciendo los elementos de abajo

Estructura de un Programa

Un programa en *BLA* consiste en una secuencia de declaraciones, donde cada declaración establece una variable o función. En el caso de las funciones, las declaraciones deben incluir el cuerpo de éstas. Para las variables, puede o no incluir su inicialización.

Todo programa debe tener una función denominada `main` con retorno `none`. Esta función sirve de punto de entrada a la ejecución del programa. `main` puede no tomar argumentos o recibir un arreglo de cadena de caracteres, correspondiente a los parámetros con los que se invocó al programa.

Alcance

BLA soporta niveles anidados de alcance. El alcance es estático. Una declaración en el nivel más alto tiene alcance global. Cada función tiene su propio alcance local para sus parámetros y otras variables locales que pueden ser declaradas en el cuerpo de estas. Un par de llaves de la forma `{ ... }` define un nuevo alcance anidado dentro de un alcance local. Los alcances locales opacan u ocultan los alcances más externos, es decir, una variable `a` definida en una función oculta cualquier otra variable con el mismo nombre definida en un alcance externo, por ejemplo, una variable `a` en alcance global.

- Todos los identificadores deben ser declarados.
- Una variable es visible posterior a su declaración. Es decir, no puede ser utilizada como *lvalue* ni *rvalue* previamente a su declaración.
- Las funciones son visibles desde cualquier punto del programa, incluso previo a su declaración.
- Los identificadores para variables dentro de un mismo alcance deben ser únicos. Igualmente, una variable no puede tener el mismo nombre que una función.
- Dos funciones pueden tener el mismo identificador, siempre que sus firmas sean distintas, esto es, que su lista de argumentos tengan aridad o tipos distintos. Por ejemplo, las primeras tres funciones se consideran distintas, mientras que las dos últimas se consideran iguales:
  - `fun int foo(int a, float b)`
  - `fun int foo(float a, int b)`
  - `fun float foo()`
  - `fun int foo(int a, int b)`
  - `fun int foo(int b, int a)`
- Las declaraciones en alcance global son accesibles a lo largo de todo el programa para funciones y posterior a su declaración para variables, a menos que sean ocultadas por variables del mismo nombre en alcances internos.
- Las declaraciones en alcances cerrados no son accesibles.

Tipos

Los tipos nativos del lenguaje son `int`, `float`, `boolean`, `string` y `none`.

Los arreglos pueden ser multi-dimensionales y de cualquier tipo distinto a `none`.

BLA permite crear nuevos tipos de datos.

Variables

Las variables pueden ser declaradas de cualquier tipo no vacío, es decir, de cualquier tipo distinto a `none`. Esto incluye arreglos y tipos definidos por el usuario. Las variables declaradas fuera de alcances locales tienen por defecto alcance global.

Las variables declaradas en la lista de parámetros formales o cuerpo de la función tienen alcance local.

Las variables globales pueden ser inicializadas con expresiones complejas siempre que éstas no incluyan llamadas a funciones definidas por el usuario.

Arreglos

En *BLA* los arreglos son homogéneos, indexados linealmente y de tamaño constante.

- Los arreglos pueden ser de cualquier tipo primitivo distinto a `none` o tipo declarado por el usuario.
- Pueden establecerse arreglos de varias dimensiones.
- Los arreglos son inicializados por defecto con el valor 0 para los tipos `int` y `float`, y `false` para los `boolean`. Los arreglos de registros son inicializados rellenando con ceros los bytes que esta contenga.
- El número de elementos en el arreglo no puede cambiarse una vez declarado.
- Los arreglos son indexados desde cero hasta una posición antes de su longitud, a menos que es especifique lo contrario. Es decir, si el rango de un arreglo va de `a` a `b`, se puede acceder a cualquier posición mayor o igual a `a` y estrictamente inferior a `b`.
- El índice utilizado en la selección debe ser de tipo entero.
- Ocurre un error a tiempo de ejecución si se intenta acceder a una posición fuera de los límites de un arreglo.
- Los arreglos pueden ser pasados a las funciones por referencia.
- La asignación de arreglos es profunda, es decir, se copia elemento a elemento. Pueden especificarse que segmentos de arreglo se desean copiar (`slice`). Sin embargo, ambos segmentos deben tener el mismo tamaño. Si no se indica el segmento de arreglo a copiar se considera que debe ser el arreglo completo.

Strings

Se pueden incluir secuencias de caracteres constantes. Los strings son manejados como cadenas de caracteres, por lo que se puede modificar partes de este, siempre que no su tamaño no varíe.

- La asignación de strings es profunda, dado que se manejan como arreglos.
- Los strings pueden ser pasados como parámetros a las funciones por referencia.
- La comparación de strings con `==` y `!=` compara las secuencias caracter por caracter.

Funciones

La declaración de una función incluye el nombre de esta y su firma asociada, es decir, tipo de retorno y número y tipo de parámetros formales.

- Las funciones tienen alcance global, no se permiten funciones anidadas.
- Las funciones pueden tener cero o más parámetros.
- Los parámetros formales pueden ser de cualquier tipo distinto a `none`, arreglos o tipos definidos por el usuario.
- Los identificadores utilizados en los parámetros formales deben ser distintos.
- Los arreglos y tipos estructurados se pasan por referencia.
- Los parámetros formales se encuentran en alcance local, por lo que no es posible definir una variable con el mismo identificador que un parámetro formal en el primer nivel de anidamiento de una función. Sin embargo, en niveles de anidamiento mayor es posible la redefinición.
- Los tipos de retorno pueden ser únicamente tipos primitivos escalares del lenguaje. En caso de querer retornar un tipo compuesto, este debe ser pasado por referencia a la función.
- Si el tipo de retorno de una función es `none`, no es posible colocar en su cuerpo retornos con parámetros, es decir, sólo es posible colocar `return` de vacíos.
- Se permiten funciones recursivas directa o indirectamente.
- El cuerpo de las funciones debe estar contenido en un bloque de anidamiento `{ ... }`. Dicho bloque puede a su vez contener un número arbitrario de instrucciones, así como nuevos niveles de anidamiento.

Invocación de funciones

La invocación de funciones incluye el pasaje de argumentos del llamador al llamado, la ejecución del cuerpo del llamado y el retorno del control al llamador, posiblemente con un resultado. Los tipos primitivos son pasados por defecto por valor mientras que los tipos estructurados son pasados por referencia. Es posible pasar un tipo básico por referencia anteponiendo a su identificador en el parámetro formal el caracter `&`.

- El número de argumentos pasados a una función en su invocación debe coincidir con el número de parámetros formales.
- El tipo de cada argumento real en la invocación de una función debe coincidir con el tipo de cada parámetro formal.
- Los parámetros son evaluados de izquierda a derecha.
- Una función retorna el control al llamador una vez alcanzado el final de su cuerpo o al alcanzar una instrucción `return`. Si el tipo de retorno es no vacío la función debe incluir un retorno por cada posible punto de salida.

Tipos compuestos

Los tipos compuestos se declaran usando `alias` *nombre tipo*. *nombre* es la representación corta que se puede usar en declaraciones de variables que se deseen de tipo *tipo*.

- Los tipos compuestos sólo son declarables en alcance global.
- Un tipo compuesto puede ser usado en cualquier lugar donde se declare una variable.

Equivalencia de Tipos, Compatibilidad

BLA es un lenguaje fuertemente tipado. La equivalencia de tipos se da por nombre. Por lo tanto, dos tipos definidos por el usuario equivalente si y sólo si tienen el mismo nombre, no serán equivalentes si tienen distintos nombres e iguales estructuras.

Todas las operaciones se deben ejecutar con argumentos del mismo tipo, en caso de necesitar realizar conversiones estas deben ser declaradas explícitamente, de la forma `T(A)`, donde `T` es el tipo destino y `A` es el elemento original.

Asignación

BLA efectúa copia por valor tanto para los tipos básicos como para los compuestos. La instrucción `LValue =`RValue‘ copia el valor del lado derecho en la dirección indicada en el lado izquierdo.

- La asignación de tipos no primitivos es profunda.
- Un `LValue` debe ser una variable o una posición de arreglo. No es permitido utilizar constantes, exceptuanto en la asignación de inicialización.
- Ambos lados de la asignación deben tener el mismo tipo.
- Los parámetros formales de una función son considerados variables a menos que se especifique lo contrario con el uso de la palabra reservada `const` previo a su tipo.

Estructuras de Control

Las estructuras de control de *BLA* están basadas en las de *Python*.

- Una cláusula `else` o `elif` siempre se asocia al `if` más cercano aún sin cerrar.
- Las expresiones condicionales de las declaraciones `if`, `elif`, `while` y `for` deben ser de tipo `boolean`.
- Una declaración de tipo `break` o `continue` sólo pueden aparecer en el cuerpo de ciclos `while` o `for`.
- La variable de iteración en un ciclo acotado `for` no puede ser utilizada como `Lvalue`, así como no puede ser pasada por referencia en funciones que se encuentren en el cuerpo del ciclo.
- El valor de una declaración `return` (en caso de ser funciones que retornen algún tipo no vacío) deben ser del mismo tipo especificado en la firma de la función.
- Las cláusulas `if`, `elif`, `else` deben ser precedidas por una instrucción de tipo bloque. Esta puede contener a su vez más instrucciones

Expresiones

Por simplicidad, *BLA* no permite la conversión automática de tipos. Esto significa que aún cuando la expresión equivalga entre los tipos, no es posible la interacción entre ellos.

- Las constantes evalúan a su valor interno (true, false, enteros, flotantes, cadenas, etc.).
- Ambos operandos en las operaciones aritméticas (+, -, *****, / y %) deben ser ambos enteros o flotantes, pero del mismo tipo.
- El operador unario - aplica sobre enteros y flotantes y devuelve el mismo tipo del operando.
- Ambos operandos en las comparaciones aritméticas (<, >, <= y >=) deben ser enteros o flotantes, pero del mismo tipo. El resultado de la operación es `boolean`.
- Ambos operandos deben ser del mismo tipo para las operaciones de igualdad (==, !=). Al comparar dos arreglos, se hará de forma profunda. El resultado de la operación será de tipo `boolean`.
- Los operandos para las expresiones booleanas (!, &&, ||, ==> y <==) aceptan sólo argumentos de tipo `boolean` y devuelven `boolean`.
- Las operaciones booleanas || y && se evalúan completamente. No se hace corto circuito.

Librería de Funciones

Algunas funciones van a estar predefinidas y van a poder usarse sin necesidad de declararlas o de importar alguna librería explícitamente.

Verificaciones a Tiempo de Ejecución

Algunas verificaciones no pueden conocerse estáticamente y deben esperar hasta la ejecución para indicar la falla:

1. El acceso a índices fuera de la definición de un arreglo terminará la ejecución del programa.

TODO

- Agregar a la definición del lenguaje: *BLA* es un lenguaje especialmente diseñado para facilitar la labor de los programadores en problemas de programación dinámica, donde sea de gran utilidad la memorización de valores de retorno de funciones.
- conversión automática de strings a enteros (por ejemplo leídos de entrada)
- tamaño de arreglo conocido en tiempo de ejecución

- comenzar con una sóla firma por función y luego expandirla a múltiples
- agregar parámetros opcionasles al final de la firma. esto implica que la firma quedaría definida por los parámetros obligatorios
- las instrucciones if/elif deberían comenzarse usando llaves y luego dar la opción de no
- la función debe ser un tipo de bloque. esto para que la declaración de variables no se pueda sobrescribir
- implementar `fuzzy_boolean`
- implementar expresiones booleanas con corto circuito.
- asignación de subconjuntos de arreglos
- foreach
- las referencias se van a tratar originalmente como si fueran la misma variable luego se van a hacer apuntadores
- se permiten instrucciones vacías
- ejemplo de instrucción vacía
- definir tipo unión y registro en 143
- cuál va a ser el ancho máximo de este documento?
- agregar memorizacion
- Se permite la sobrecarga de funciones siempre que las firmas de éstas no coincidan (en número de argumentos o en sus tipos).