The Madafaka Language

Como proyecto para cadena de electivas de especialización en Lenguajes de programación, se ha desarrollado un lenguaje de programación llamado **Madafaka**. Madafaka es un lenguaje imperativo, fuertemente tipado, con soporte para estructuras de datos anidadas. Está inspirado en C/C++ con algunas variantes, algunas por cuestiones de disminuir la complejidad del proyecto (y que sea factible terminarlo en 6 meses) y otras para agregar un toque *entretenido* al lenguaje.

A continuación se especifica la sintaxis y semántica del lenguaje, con el objetivo de orientar a los programadores en el uso adecuado de Madafaka.

Elementos léxicos

Madafaka posee las siguientes palabras reservadas, que no pueden ser usadas en ningún programa como identificadores:

```
mada faka var fordafak ifdafak whiledafak unidafak wdafak rdafak
vdafak idafak fdafak cdafak sdafak strdafak true false
```

Un identificador es una secuencia arbitraria de números y letras, que comienza por una letra. Los identificadores de variables son *sensibles* a las mayúsculas y minúsculas, así que `hola` es un identificador, a la vez que `Hola` es otro distinto, mientras que `Mada` es otro diferente

Una constante entera es una secuencia arbitraria de dígitos numéricos, expresada en base 10 *únicamente*, como `42`, `127` y `50`. A su vez, una constante en punto flotante en una secuencia de números, con un punto delimitando la parte entera de la decimal, como `42.0` o `3.1415`. Las constantes booleanas son `true` y `false`.

Se permite la elaboración de expresiones aritméticas o booleanas, utilizando los símbolos siguientes:

```
( ) - == * / % + > >= < <=
```

Estas expresiones pueden ser asignadas a las variables con el símbolo `=`.

Exceptuando como separador de las palabras, los espacios son ignorados por el lenguaje. Asimismo, cualquier cadena de caracteres que empiece por `??` es ignorada, y corresponde a los comentarios. No se admiten comentarios multilínea

Estructura de un programa Madafaka

Un programa de Madafaka consiste en un bloque global, delimitado por las palabras reservadas `mada` y `faka`, que inicia con una serie de declaraciones de variables, delimitadas por dos líneas con @@ y con ; al final de *cada* declaración, y continúa con una serie de instrucciones, siempre con ; al final de *cada* instrucción, potencialmente vacía. Por simplicidad de implementación, las declaraciones de variables deben ir *antes* que cualquier instrucción. Adicionalmente, en un cuerpo de un bloque se puede crear otro bloque mediante las palabras `mada` y `faka`, permitiéndose el anidamiento de bloques. Un programa de Madafaka tiene la siguiente estructura:

```
mada
    @@
    declaracion1;
    declaracion2;
    ...
    declaracionN
    @@

    instruccion1;
    instruccion2;
faka
```

Alcance

Madafaka soporta varios niveles de alcance, siempre de tipo **estático**. Se inicia con un alcance global,en el que se agregan todas las declaraciones de variables que se hacen al inicio del bloque principal. Adicionalmente, cada declaración de función que se escribe designa un nuevo alcance, en el que se insertan las variables declaradas en esa función. El mismo comportamiento aplica cuando se crean bloques arbitrarios. En todo momento aplican las siguientes reglas:

- Todos los identificadores deben ser declarados.
- Los identificadores declarados en un bloque deben ser todos únicos. No se aceptan declaraciones repetidas en el mismo bloque, aún cuando sean del *mismo tipo*
- Cuando se hace una declaración en un bloque, cuando ya existe una declaración para ese mismo nombre en un bloque circundante, la primera *oculta* a esta última.
- Las declaraciones en el bloque global son accesibles en todos los bloques

Variables y Tipos de datos

Madafaka incluye inicialmente los tipos de datos entero, flotante, string, y caracter, que llevan de nombre idafak, fdafak, sdafak, y cdafak respectivamente. Para hacer una declaración de variables, se utiliza la sintaxis

```
<TipoDeDato> <identificador>;
```

En adición a esto, se permite la construcción de tipos compuestos (struct, que recibe el nombre de strdafak en Madafaka, o unions, que reciben el nombre de unidafak en Madafaka) al estilo C/C++. Para esto se utiliza la sintaxis

```
strdafak <identificador> mada
    <TipoDeDatoCampo1> <identificador>;
    <TipoDeDatoCampo2> <identificador>;
    ...
faka;
```

Arreglos

Los arreglos en Madafaka son unidimensionales, homogéneos, y estáticos. Son declarados de la siguiente manera:

```
<TipoDeDato> <identificador>[tamaño1];
```

Donde `tamañoN` consiste en una constante entera que determina una de las dimensiones del arreglo.

- Los índices del arreglo van desde 0 a tamañoN-1

Strings

En Madafaka, los strings pueden utilizarse para asignar a las variables, para imprimirse en pantalla y para compararse en expresiones booleanas. Hasta el momento no se han implementado más operaciones con strings

Declaraciones e invocaciones de funciones

Las declaraciones de funciones se realizan mediante la sintaxis

```
<TipoRetorno> <Identificador>(<TipoArg1> <id1>, <TipoArg2> <id2>,...)
```

- Una función puede tener cero o más parámetros.
- Los argumentos pueden ser de todos los tipos soportados por el lenguaje: tipos primitivos, compuestos o arreglos.
- Las funciones pueden declararse en cualquier parte de los bloques de declaraciones
- Por defecto el pasaje de parámetros es **por valor**. Sin embargo puede utilizarse el pasaje por referencia colocando la palabra `var` antes de la declaración del parámetro.
- No se permite la sobrecarga de funciones.
- Se permiten funciones recursivas.
- Al llamar una función, deben pasarse la misma cantidad de parámetros que en la declaración
- Puede utilizarse una función como valor en una expresión, siempre que el valor de retorno sea del **tipo adecuado**

Asignaciones

En Madafaka, las asignaciones de tipos primitivos y compuestos se hacen con el modelo de valor: el valor del lado derecho se almacena en la dirección de memoria referenciada por el lado izquierdo. En cuanto a arreglos, se utiliza el modelo de referencia: la referencia del objeto del lado derecho se utiliza para actualizar el valor apuntado por el lado izquierdo. Estas decisiones se encuentran sujetas a cambios.

Expresiones

Por simplicidad, no se soporta la conversión implícita de tipos en una expresión. Sin embargo, el resto de las operaciones típicas está soportado:

- En las comparaciones aritméticas, ambos lados de la comparación deben ser del mismo tipo (enteros con enteros, flotantes con flotantes)

- Al hacer operaciones aritméticas, ambos lados de la operación pueden ser del mismo tipo (enteros con enteros, flotantes con flotantes), pero también se acepta operaciones aritméticas entre enteros y flotantes a través del **ensanchamiento** de tipos (una suma de enteros con flotantes produce un flotante). Lo importante es que las asignaciones se realicen respetando el sistema de tipos (una variable flotante no puede asignarse en un tipo entero)
- Las expresiones booleanas no soportan cortocircuitos
- Las comparaciones de igualdad (`==`) deben hacerse entre tipos equivalentes
- Al hacer expresiones booleanas con los operadores `and` y `or`, ambos lados de la operación deben ser booleanos
- Se soportan constantes en las expresiones como `42`, `3.7`, y `true`.

Estructuras de control

Al igual que C/C++, Madafaka posee varias estructuras de control de utilidad:

Selección

El bloque puede contener un bloque sencillo, así como una estructura if-then-else

```
ifdafak <ExpresionBooleana> mada
    <inst1>
    <inst2>
    ...
faka;

ifdafak <ExpresionBooleana> mada
    <inst1>
    <inst2>
    ...
faka
else mada
    <inst1>
    <inst2>
    ...
faka;
```

Iteración controlada:

Tiene una sintaxis muy similar a la de C/C++, aunque con algunas diferencias sencillas. La variable del ciclo debe estar declarada en el bloque actual o en algún bloque circundante, al igual que cada variable en la condición del bloque

```
fordafak(<AsignaciónInicial>;<CondiciónDeBloque>;<InstrucciónDeIncremento>) mada
    <inst1>
    <inst2>
    ...
faka;
```

Iteración sencilla:

```
whiledafak <CondiciónBooleana> mada
    <inst1>
    <inst2>
    ...
    madabreak;
faka;
```

Librería estándar

Con miras en la simplicidad, Madafaka ofrece una cantidad reducida de instrucciones de librería estándar, enfocadas principalmente en Input/Output

- `wdafak` <`Expresión`> coloca en la salida estándar el resultado de evaluar la expresión.
- `rdafak` <`Identificador`> obtiene un valor de la entrada estándar y lo almacena en el Lvalue especificado, que puede ser una variable, un campo de un Struct o una posición de un arreglo.