Trabajo Práctico Obligatorio
Diseño de Compiladores
2014

Objetivos

El objetivo de este trabajo obligatorio es la construcción de un intérprete de un subconjunto del lenguaje Javascript. En este intérprete NO nos interesa trabajar con los aspectos relacionados con la web del lenguaje Javascript

Sintaxis

Las normas básicas que definen la sintaxis de JavaScript son las siguientes:

- No se tienen en cuenta los espacios en blanco y las nuevas líneas: el intérprete de JavaScript ignora cualquier espacio en blanco sobrante, por lo que el código se puede ordenar de forma adecuada para entenderlo mejor (tabulando las líneas, añadiendo espacios, creando nuevas líneas, etc.)

- Se distinguen las mayúsculas y minúsculas: El lenguaje es sensible a mayúsculas y minúsculas. La variable **suma** no es igual a la variable **SuMa**

- No se define el tipo de las variables: al crear una variable, no es necesario indicar el tipo de dato que almacenará. De esta forma, una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script.

- Cada sentencia debe terminar con el carácter de punto y coma (;)

- Se pueden incluir comentarios: los comentarios se utilizan para añadir información en el código fuente del programa. JavaScript define dos tipos de comentarios, los de una sola línea y los que ocupan varias líneas.

  Ejemplo de comentario de una sola línea:

```
// a continuación se muestra un mensaje
alert("mensaje de prueba");
```

  Ejemplo de comentario de varias líneas:

```
/* Los comentarios de varias líneas son muy útiles
cuando se necesita incluir bastante información
en los comentarios */
alert("mensaje de prueba");
```

Variables

Las variables en JavaScript se crean mediante la palabra reservada **var**.

```
var numero_1 = 3;
var numero_2 = 1;
var resultado = numero_1 + numero_2;
```

La palabra reservada var solamente se debe indicar al declarar la variable. Cuando se utilizan las variables en el resto de instrucciones del script, solamente es necesario indicar su nombre.

Si cuando se declara una variable se le asigna también un valor, se dice que la variable ha sido inicializada. En JavaScript no es obligatorio inicializar las variables, ya que se pueden declarar por una parte y asignarles un valor posteriormente.

```
var numero_1;
var numero_2;

numero_1 = 3;
numero_2 = 1;

var resultado = numero_1 + numero_2;
```

Una característica interesante de JavaScript es que tampoco es necesario declarar las variables. En otras palabras, se pueden utilizar variables que no se han definido anteriormente mediante la palabra reservada var. El ejemplo anterior también es correcto en JavaScript de la siguiente forma:

```
var numero_1 = 3;
var numero_2 = 1;
resultado = numero_1 + numero_2;
```

La variable resultado no está declarada, por lo que JavaScript crea una variable global (más adelante se verán las diferencias entre variables locales y globales) y le asigna el valor correspondiente. De la misma forma, también sería correcto el siguiente código:

```
numero_1 = 3;
numero_2 = 1;
resultado = numero_1 + numero_2;
```

El nombre de una variable debe cumplir las siguientes normas:

- Sólo puede estar formado por letras, números y los símbolos $ (dólar) y _ (guión bajo).
- El primer carácter no puede ser un número.

Por tanto, las siguientes variables tienen nombres correctos:

```
var $numero1;
var _$letra;
var $$$otroNumero;
var $_a__$4;
```

Sin embargo, las siguientes variables tienen identificadores incorrectos:

```
var 1numero;        // Empieza por un número
```

```
var numero\1_123;  // Contiene un carácter "\"
```

Tipos de datos de las variables

Aunque todas las variables de JavaScript se crean de la misma forma (mediante la palabra reservada var), la forma en la que se les asigna un valor depende del tipo de valor que se quiere almacenar (números, textos, etc.)

Numéricas

Se utilizan para almacenar valores numéricos enteros o flotantes.

```
var iva = 16;         // variable tipo entero
var total = 234.65;   // variable tipo decimal
```

Texto

Se utilizan para almacenar caracteres, palabras y/o frases de texto. Para asignar el valor a la variable, se encierra el valor entre comillas dobles o simples, para delimitar su comienzo y su final:

```
var mensaje = "Bienvenido a nuestro sitio web";
var nombreProducto = 'Producto ABC';
var letraSeleccionada = 'c';
```

Si por ejemplo el texto contiene comillas simples o dobles, la estrategia que se sigue es la de encerrar el texto con las comillas (simples o dobles) que no utilice el texto:

```
var texto1 = "Una frase con 'comillas simples' dentro";
var texto2 = 'Una frase con "comillas dobles" dentro';
```

No obstante, a veces las cadenas de texto contienen tanto comillas simples como dobles. Para resolver estos problemas, JavaScript define un mecanismo de escape para incluir los caracteres especiales dentro del texto:

```
Una nueva línea          \n
Un tabulador             \t
Una comilla simple       \'
Una comilla doble        \"
Una barra inclinada      \\
```

Arrays

Un array es una colección de variables, que pueden ser todas del mismo tipo o cada una de un tipo diferente:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes",
"Sábado", "Domingo"];
```

Para definir un array, se utilizan los caracteres [y] para delimitar su comienzo y su final y se utiliza el carácter , (coma) para separar sus elementos:

```
var nombre_array = [valor1, valor2, ..., valorN];
```

Cada elemento se accede indicando su posición dentro del array.

```
var diaSeleccionado = dias[0];     // diaSeleccionado = "Lunes"
var otroDia = dias[5];             // otroDia = "Sábado"
```

Las posiciones se empiezan a contar en el 0, por lo que el primer elemento ocupa la posición 0 y se accede mediante dias[0].

Booleanos

Una variable de tipo boolean solamente puede tomar dos valores: true (verdadero) o false (falso). No se puede utilizar para almacenar números y tampoco permite guardar cadenas de texto.

```
var clienteRegistrado = false;
var ivaIncluido = true;
```

Operadores

Se deberán soportar los siguientes operadores:

- Asignación (=)
- Incremento y decremento (++ y --)
- Lógicos (&&, || y !)
- Matemáticos (+, -, * y /)
- Relacionales (>, >=, <, <=, == y !=)

Los operadores deberán poder ser aplicados sobre los tipos de datos tradicionales.

Estructuras de control

Se deberán soportar las siguientes estructuras de control:

- **Bloque:** Cualquier conjunto de sentencias dentro de dos llaves { … }.

```
{
  ...
  ...
}
```

- **If:** Decisión con una opción

```
if (condición-booleana) {
  ...
}
```

- **If / Else:** Decisión con dos opciones

```
if (condición-booleana) {
  ...
} else {
  ...
}
```

- **For:** Iteración en base a una condición

```
for(inicializacion; condicion; actualizacion) {
  ...
}

var mensaje = "Hola, estoy dentro de un bucle";

for (var i = 0; i < 5; i++) {
  alert(mensaje);
}
```

- **Break y Continue:** Se soportan las sentencias para salir de la iteración actual (break) y saltar a la próxima iteración del ciclo (continue)

Estas estructuras de control son los bloques básicos de construcción de cualquier programa. Su sintaxis es muy similar entre los diferentes lenguajes de propósito general existentes.

Propiedades y funciones utiles

JavaScript incorpora una serie de herramientas y utilidades para el manejo de las variables. De esta forma, muchas de las operaciones básicas con las variables, se pueden realizar directamente con las utilidades que ofrece JavaScript.

Funciones para manipulación de texto

- length, calcula la longitud de una cadena de texto (el número de caracteres que la forman)

```
var mensaje = "Hola Mundo";
var numeroLetras = mensaje.length; // numeroLetras = 10
```

- concat(), permite concatenar dos cadenas

```
var mensaje1 = "Hola";
var mensaje2 = mensaje1.concat(" Mundo"); // mensaje2 = "Hola
Mundo"
```

- toUpperCase, toLowerCase(), cambian el case de la cadena

```
var mensaje1 = "Hola";
var mensaje2 = mensaje1.toUpperCase(); // mensaje2 = "HOLA"
var mensaje1 = "HolA";
var mensaje2 = mensaje1.toLowerCase(); // mensaje2 = "hola"
```

- charAt(posicion) e indexOf(caracter), permiten obtener el carácter en una posición y buscar la posición de un carácter en la cadena

```
var mensaje = "Hola";
var letra = mensaje.charAt(0); // letra = H
letra = mensaje.charAt(2);     // letra = l
```

```
var mensaje = "Hola";
var posicion = mensaje.indexOf('a'); // posicion = 3
posicion = mensaje.indexOf('b');     // posicion = -1
```

- lastIndexOf(caracter), similar a indexOf, pero devuelve la última posición.

- substring(inicio, final), retorna el substring desde la posición de inicio, hasta la posición final, sin considerar el carácter en la posición final.

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(2); // porcion = "la Mundo"
porcion = mensaje.substring(5);     // porcion = "Mundo"
porcion = mensaje.substring(7);     // porcion = "ndo"

var mensaje = "Hola Mundo";
var porcion = mensaje.substring(-2); // porcion = "Hola Mundo"

var mensaje = "Hola Mundo";
var porcion = mensaje.substring(1, 8); // porcion = "ola Mun"
porcion = mensaje.substring(3, 4);     // porcion = "a
```

- split(separador), convierte una cadena, en un array de cadenas

```
var mensaje = "Hola Mundo, soy una cadena de texto!";
var palabras = mensaje.split(" ");
// palabras = ["Hola", "Mundo,", "soy", "una", "cadena", "de",
"texto!"];

var palabra = "Hola";
var letras = palabra.split(""); // letras = ["H", "o", "l", "a"]
```

Funciones para arrays

- length, devuelve la cantidad de elementos de un array

```
var vocales = ["a", "e", "i", "o", "u"];
var numeroVocales = vocales.length; // numeroVocales = 5
```

- concat(), permite concatenar los elementos de un array

```
var array1 = [1, 2, 3];
array2 = array1.concat(4, 5, 6);   // array2 = [1, 2, 3, 4, 5,
6]
array3 = array1.concat([4, 5, 6]); // array3 = [1, 2, 3, 4, 5,
6]
```

- join(separador), permite juntar los elementos de un array en un string

```
var array = ["hola", "mundo"];
var mensaje = array.join(""); // mensaje = "holamundo"
mensaje = array.join(" ");    // mensaje = "hola mundo"
```

- pop(), permite eliminar el último elemento de un array

```
var array = [1, 2, 3];
var ultimo = array.pop();
// ahora array = [1, 2], ultimo = 3
```

- push(), agrega un elemento al final del array

```
var array = [1, 2, 3];
array.push(4);
// ahora array = [1, 2, 3, 4]
```

- shift(), permite eliminar el primer elemento de un array

```
var array = [1, 2, 3];
var primero = array.shift();
// ahora array = [2, 3], primero = 1
```

- reverse(), permite invertir los elementos de un array

```
var array = [1, 2, 3];
array.reverse();
// ahora array = [3, 2, 1]
```

Funciones para números

- NaN: Es un valor que representa un resultado numérico no representable. Por ejemplo, para divisiones por 0

- isNaN(valor): Permite testear si el valor indicado “valor” es un valor no numérico. Debería dar verdadero solo en el caso que valor sea NaN

- parse(valor): Permite convertir un valor de tipo string en un número.

```
var numeroStr = “1234”;
var numero = parse(numeroStr);
// ahora numero contiene el numero 1234
```

Funciones

Para trabajar con funciones se utilizaran los mecanismos tradicionales de los lenguajes de programación de propósito general. Tendremos una definición de función, con una lista posible de argumentos, y un posible valor de retorno. Una definición típica de función, con su posterior invocación es de la forma:

```
// Definición de la función
function suma_y_muestra(primerNumero, segundoNumero) {
  var resultado = primerNumero + segundoNumero;
  alert("El resultado es " + resultado);
}

// Declaración de las variables
var numero1 = 3;
var numero2 = 5;
```

```
// Llamada a la función
suma_y_muestra(numero1, numero2);
```

En el caso que se devuelva un valor, se usa la sentencia return. El formato de la llamada es:

```
function calculaPrecioTotal(precio, porcentajeImpuestos) {
  var gastosEnvio = 10;
  var precioConImpuestos = (1 + porcentajeImpuestos/100) * precio;
  var precioTotal = precioConImpuestos + gastosEnvio;
  return precioTotal;
}

var precioTotal = calculaPrecioTotal(23.34, 16);
var otroPrecioTotal = calculaPrecioTotal(15.20, 4);
```

En caso de que no se quiera pasar parámetros a una función, basta con dejar la lista de parámetros vacía. **Las funciones, también soportan llamadas recursivas.**

Ámbito de las variables

El ámbito de una variable (llamado "scope" en inglés) es la zona del programa en la que se define la variable. JavaScript define dos ámbitos para las variables: global y local.

El siguiente ejemplo ilustra el comportamiento de los ámbitos:

```
function creaMensaje() {
  var mensaje = "Mensaje de prueba";
}
creaMensaje();
alert(mensaje);
```

El ejemplo anterior define en primer lugar una función llamada creaMensaje que crea una variable llamada mensaje. A continuación, se ejecuta la función mediante la llamada creaMensaje(); y seguidamente, se muestra mediante la función alert() el valor de una variable llamada mensaje.

Sin embargo, al ejecutar el código anterior no se muestra ningún mensaje por pantalla. La razón es que la variable mensaje se ha definido dentro de la función creaMensaje() y por tanto, es una variable local que solamente está definida dentro de la función.

Cualquier instrucción que se encuentre dentro de la función puede hacer uso de esa variable, pero todas las instrucciones que se encuentren en otras funciones o fuera de cualquier función no tendrán definida la variable mensaje. De esta forma, para mostrar el mensaje en el código anterior, la función alert() debe llamarse desde dentro de la función creaMensaje():

```
function creaMensaje() {
  var mensaje = "Mensaje de prueba";
  alert(mensaje);
}
creaMensaje();
```

Además de variables locales, también existe el concepto de variable global, que está definida en cualquier punto del programa (incluso dentro de cualquier función).

```
var mensaje = "Mensaje de prueba";
function muestraMensaje() {
  alert(mensaje);
}
```

El código anterior es el ejemplo inverso al mostrado anteriormente. Dentro de la función muestraMensaje() se quiere hacer uso de una variable llamada mensaje y que no ha sido definida dentro de la propia función. Sin embargo, si se ejecuta el código anterior, sí que se muestra el mensaje definido por la variable mensaje.

El motivo es que en el código JavaScript anterior, la variable mensaje se ha definido fuera de cualquier función. **Este tipo de variables automáticamente se transforman en variables globales y están disponibles en cualquier punto del programa (incluso dentro de cualquier función)**.

De esta forma, aunque en el interior de la función no se ha definido ninguna variable llamada mensaje, la variable global creada anteriormente permite que la instrucción alert() dentro de la función muestre el mensaje correctamente.

Si una variable se declara fuera de cualquier función, automáticamente se transforma en variable global independientemente de si se define utilizando la palabra reservada var o no. Sin embargo, las variables definidas dentro de una función pueden ser globales o locales.

Si en el interior de una función, las variables se declaran mediante **var** se consideran locales y las variables que no se han declarado mediante **var**, se transforman automáticamente en variables globales.

Por lo tanto, se puede rehacer el código del primer ejemplo para que muestre el mensaje correctamente. Para ello, simplemente se debe definir la variable dentro de la función sin la palabra reservada var, para que se transforme en una variable global:

```
function creaMensaje() {
  mensaje = "Mensaje de prueba";
}

creaMensaje();
alert(mensaje);
```

¿Qué sucede si una función define una variable local con el mismo nombre que una variable global que ya existe? En este caso, las variables locales prevalecen sobre las globales, pero sólo dentro de la función:

```
var mensaje = "gana la de fuera";
function muestraMensaje() {
  var mensaje = "gana la de dentro";
  alert(mensaje);
```

```
}
alert(mensaje);
muestraMensaje();
alert(mensaje);
```

El código anterior muestra por pantalla los siguientes mensajes:

```
gana la de fuera

gana la de dentro

gana la de fuera
```

Estructura de un programa Javascript

Los programas procesados por este intérprete estarán siempre incluidos en un único archivo. La estructura del archivo será la siguiente:

- Definición de variables globales: Consiste en una serie de definiciones de variables globales
- Definición de funciones: Consiste en una lista de definiciones de funciones
- Código Javascript: Consiste en una sección de código Javascript. Este código será lo primero que se ejecute cuando el intérprete comience.

Cualquiera de las tres secciones anteriores puede estar vacía.

Por ejemplo:

```
var mensaje1 = "Mensaje 1";
var mensaje2 = "Mensaje 2";
var mensaje3 = "Mensaje 3";

function muestraMensaje1() {
   alert(mensaje1);
}

function muestraMensaje2() {
   alert(mensaje2);
}

function muestraMensaje3() {
   alert(mensaje3);
}

// Comienza el programa principal
muestraMensaje1();
muestraMensaje2();
muestraMensaje3();
```

Ejecución de los programas

El mecanismo de ejecución de los programas Javascript será a través de la línea de comando. Se indicara la ruta del archivo con el programa Javascript a correr, junto con los parámetros que estos pudieran necesitar. Por ejemplo:

- `java package_clase_interprete nombre_archivo.js XXX YYY ZZZ`

En este caso, XXX, YYY y ZZZ representan tres parámetros que se le pasan a este programa para que ejecute. Dentro del programa, estos parámetros podrán ser accedidos a través del array $ARG. Este array deberá ser cargado antes de comenzar a ejecutar el código del programa. En este caso, $ARG[0] contiene XXX, $ARG[1] contiene YYY y $ARG[2] contiene ZZZ.

NULL

Existe un valor especial, asignable a cualquier variable. Este valor es NULL, y como en los lenguajes tradicionales orientados a objetos, representa la ausencia de valor.

Tratamiento de errores

El requerimiento mínimo es terminar el proceso de interpretación una vez que se detecta un error indicando el número de línea y de ser posible el tipo de error (sintáctico, de tipo, etc.).

Son aceptables soluciones en los que los mensajes de error son de la forma:

- `Error sintáctico en línea 2 cerca de "var x = 10"`
- `Tipo de datos incorrecto en sentencia IF en línea 10`

Mecanismos para el manejo de error más elaborados, serán tenidos en cuenta, pero se sugiere comenzar con un manejo mínimo del error, y luego expandirlo si se tiene tiempo disponible.

Ambiente de ejecución

Deberá implementar un ambiente de ejecución apropiado para permitir ejecutar los programas Javascript que se puedan construir según este obligatorio. Entre otras cosas, este ambiente de ejecución regulara el llamado a funciones, la gestión de memoria, manejo de variables, registro de activación, etc.

Objetivos, formas y plazos de entrega

Algunos aspectos formales a tener en cuenta…

Sobre el objetivo

El objetivo principal del obligatorio es la entrega de un intérprete de Javascript, **QUE FUNCIONE**.

Las características del lenguaje que deben ser tenidas en cuenta para dicho intérprete son las presentadas en este documento. Toda otra característica que no esté en este documento, se

considera opcional (a menos que sea aclarada su obligatoriedad en la página web del curso o a través del news)

Sobre la forma de trabajo

El trabajo deberá ser realizado en grupos de hasta tres estudiantes. Los mismos deberán ser informados antes del 9/6 al docente encargado del curso. El día 10/6 será informada la lista final de grupos del obligatorio. Se realizaran clases de consulta los días miércoles hasta la fecha de entrega, en el horario de 8:30 a 9:30, en el salón 301.

IMPORTANTE: Una vez formado un grupo, no se aceptaran separaciones o divisiones en dicho grupo. En caso de darse esta situación, TODOS los integrantes del curso pierden el obligatorio. En caso de que parte de los integrantes abandonen, solo a estos se les considera como no aprobado el curso.

Sobre la plataforma de trabajo

El trabajo deberá ser realizado utilizando el lenguaje Java. Se deberán utilizar los utilitarios JFLEX y CUP para realizar el análisis léxico y sintáctico respectivamente. En caso de ser necesario, se podrá utilizar código descargado de Internet (debidamente documentado), siempre y cuando este NO RESUELVA una parte fundamental del problema planteado. Por ejemplo, no sería aceptable si se entrega un intérprete de Javascript no desarrollado por el grupo.

Las pruebas de correctitud del obligatorio serán realizadas en plataforma Windows. Puede desarrollar en cualquier plataforma que considere necesaria, pero se recomienda probar en plataforma Windows lo que se va a entregar.

MUY IMPORTANTE:

Se deberá entregar el código fuente de la solución, así como los scripts necesarios para compilar y obtener una versión funcional del intérprete. El grupo se deberá asegurar que el código fuente se puede compilar, ya que SOLO se ejecutara el intérprete obtenido de la compilación de los fuentes entregados. Al ser el obligatorio desarrollado en Java, puede utilizarse ANT o MAVEN para automatizar el proceso.

No se aceptaran soluciones que utilicen archivos propietarios de proyectos desarrollados en IDEs como Eclipse, JBoss Studio, Websphere, etc. El grupo deberá asegurarse que el código se puede compilar sin necesidad de contar con dichas IDEs

Se recomienda entonces entregar el fuente, junto con un script ANT que genere el binario del intérprete.

Sobre la entrega

La fecha de entrega final para este obligatorio es el lunes 14 de julio del 2014 (inclusive)

Se deberá entregar al docente del curso, en forma electrónica, lo siguiente:

- Documentación del obligatorio (breve) presentando los aspectos principales del diseño de la gramática, chequeos, las tablas de símbolos, la representación intermedia, tratamiento de error y el proceso de interpretación. Se espera, muy especialmente, que fundamenten adecuadamente las decisiones de diseño tomadas a lo largo del obligatorio

- Los archivos de prueba ejecutados por el grupo sobre el intérprete. En caso de las pruebas que contienen error, indicar los errores obtenidos. En caso de que haya un problema conocido con la ejecución de los programas de prueba entregados, esto deberá ser documentado apropiadamente

Sobre la evaluación

Se evaluaran los siguientes puntos:

- **Correcto funcionamiento del intérprete**, ejecutando los programas de prueba entregados por el grupo, en los cuales se prueben los aspectos del lenguaje que deben ser implementados. **SI EL INTERPRETE NO FUNCIONA CORRECTAMENTE, EL CURSO SE CONSIDERA PERDIDO PARA TODOS LOS INTEGRANTES**

- **Calidad en la implementación**. **SOLO SI EL INTERPRETE FUNCIONA CORRECTAMENTE**, se evaluaran como puntos extra, la calidad con que sea haya implementado la solución, las características extra que se le hayan agregado al intérprete, las mejoras al manejo e informe de errores, etc.

- En caso de duda por parte del docente sobre la implementación del interprete, los integrantes del grupo (en su totalidad o parcialmente) podrán ser llamados a una defensa, en la cual se les puede realizar preguntas concretas sobre la implementación realizada. **EN CASO DE COMPROBARSE QUE UN MIEMBRO DEL EQUIPO NO TRABAJO EN EL OBLIGATORIO, EL CURSO SE CONSIDERA PERDIDO PARA DICHO INTEGRANTE**

- El puntaje máximo del obligatorio es de 100 puntos. Se aprueba con el 60% de los puntos. Este puntaje se promedia con el obtenido en la parte teórica del curso.

Sobre las copias

Para resolver este tema, se utilizara el mecanismo tradicional aplicado en el InCo cuando se detectan copias entre los grupos**. TODOS LOS INVOLUCRADOS PIERDEN EL CURSO** y se informa apropiadamente a la Facultad sobre esta situación.