Procesadores de Lenguaje

Práctica de la asignatura

1ª entrega

Boris Carballa Corredoira

Creando un compilador de Java

Esta entrega tiene como propósito explicar qué intrucciones tendrá el lenguaje de programación que compilará el compilador de las sucesivas entregas. El lenguaje de programación estará basado en Java, intentando tener el máximo parecido pero tratando de que sea suficientemente abarcable.

• Tipos:

Las variables deberán tiparse. Habrá los siguientes tipos básicos:

- Booleanos: tendrán *boolean* como palabra reservada. Podrá tomar dos valores: *true* y *false*. El valor por defecto será false.

- Enteros: tendrán *int* como palabra reservada. Su valor será un número entre -2147483648 y 2147483647. Su valor por defecto será el 0.

- Double: tendrán *double* como palabra reservada. Su valor será un número decimal, esto es, una serie de dígitos (números entre 0 y 9), opcionalmente con un punto (como máximo) entre todos ellos. Su valor por defecto será 0.

- Letras: tendrán *char* como palabra reservada. Su valor será una letra perteneciente a ASCII entre comillas simples. Su valor por defecto será '\0' (el carácter nulo).

Los operadores serán los siguientes, en orden de prioridad de menor a mayor (aquellos con la misma prioridad desempatarán con asociatividad por la izquierda):

Infijos:
- Lógicos: || && (solo entre booleanos)
- Igualdad: == !=
- Relacionales: < > <= >=
- Aritméticos: + -
- Aritméticos (bis): * / %

Prefijos:
- Unario: !

Postfijos:
- expresion++ expresion - -

El operador de asignación es un operador infijo con la prioridad más baja y del que hablaré en la siguiente parte.

Habrá también variables de tipo array de otros tipos, así como array de arrays (lo cual tiene sentido si consideramos el tipo array de un tipo como un nuevo tipo). Cada elemento tendrá el valor por defecto correspondiente al tipo base del array, excepto que se inicialice a un valor concreto en su declaración. El array si no se inicializa tendrá un valor por defecto de *null*.

Dos tipos se considerarán equivalentes solo si tienen el mismo nombre.
La comprobación de tipos se realizará de forma ascendente.

Podrán crearse clases, que tendrán la palabra reservada *class* y se considerarán a todos los efectos como un nuevo tipo. Los objetos creados usando estas clases seguirán las mismas reglas para su declaración que los demás tipos y su valor por defecto si no se inicializa será *null*.

• Identificadores y ámbitos de definición:

La sintaxis para declarar las variables será la siguiente: primero la palabra reservada correspondiente al nombre del tipo (el tipo de un array se escribe como el tipo de sus elementos seguido de dos corchetes: [ ]), después el nombre de la variable (caracteres alfanuméricos o guiones bajos, no pudiendo ser la primera letra un número) y, opcionalmente, después se podrá asignar su valor inicial mediante un igual seguido del valor (que deberá ser válido según el tipo de la variable a declarar):

- Si es un tipo básico, será un valor según lo especificado en el apartado anterior.

- Si es un array de un tipo, se escribirá la palabra reservada *new* seguida del tipo, dos corchetes y un entero positivo, bien como literal o como variable (en cuyo caso se comprobará solamente que es de tipo entero), entre ambos corchetes. Opcionalmente, si no se escribe el entero entre los corchetes podrá haber dos llaves a continuación con valores entre ambas (variables o literales) del tipo correspondiente, separados por comas, para inicializar el array.

- Si el tipo es una clase, también se escribirá *new* seguido del nombre de la clase y dos paréntesis (). Por lo pronto, todas las clases deberán implementar el constructor vacío.

En el caso de no dar valor inicial, la variable se inicializará al valor por defecto de su tipo.

Además, en todos los casos anteriores también se considera válido como valor una variable ya declarada y del mismo tipo.
La inicialización finalizará con un punto y coma.

Una clase se implementará de la siguiente forma: se escribirá la palabra reservada *class* seguida del nombre de la clase (caracteres alfanuméricos o guiones bajos, no pudiendo ser la primera letra un número) y dos llaves. En el interior de las llaves deberá implementarse el constructor vacío: nombre de clase, dos paréntesis y dos llaves, con el contenido del constructor entre ambas llaves.

Implementación de clases:

```
class Clase {
        Clase() {

        }
}
```

El lenguaje tendrá un ámbito estático, que permitirá tener variables con el mismo nombre en distintos bloques. Cada método será un bloque, por lo que este será el ámbito de definición de sus variables.

Ejemplos de declaración de variables:

```
var variable;
var variable = new var(...);
var[] array;
var[] array = new array[size];
var[] array = new array[]{elem1,elem2};
var[][] array = new array[size][];
var[][] array = new array[size1][size2];

int variable = 0;
boolean variable = true;
double variable = 9.99;
char variable = ‘c’;

class Adios {
     Adios() {
          Hola hola = new Hola();
     }
}
```

• Conjunto de instrucciones del lenguaje:

La instrucción condicional empezará con la palabra reservada *if* seguida de dos paréntesis y después de dos llaves. Opcionalmente, podrá ir seguido de la palabra reservada *else* y otras dos llaves. Dentro de las llaves irá el código de cada una de las ramas y dentro de los paréntesis irá una expresión que tras ser evaluada deberá resultar en un booleano. Si da *true* se ejecutará el código entre las primeras llaves y si da *false* se ejecutará, de haberlo, el de entre las segundas llaves.

Un comentario comenzará por // y todo lo escrito entre esas dos barras (incluídas) y el siguiente salto de línea se descartará. También puede comenzar por /* y finalizar por */, quedando descartado todo el contenido entre ambos (incluídas).

La instrucción de bucle será un for: comenzará por la palabra reservada *for* seguida de dos paréntesis y dos llaves. El código que se ejecutará estará entre ambas llaves. Dentro de los parentesis habrá dos puntos y coma (exactamente), que servirán de delimitadores de tres zonas: la primera será una instrucción que se ejecutará siempre y de primera, la segunda será una expresión que se evaluará a un booleano, se ejecutará al empezar cada iteración y que expresará si debe seguir ejecutándose el bucle; la tercera zona será una instrucción que se ejecutará al terminar cada iteración antes de la reevaluación de la expresión de la segunda zona.

La instrucción de asignación se acaba de explicar en el apartado anterior.
Las instrucciones anteriores y la implementación de una clase no llevan escrito un punto y coma al finalizar, todas las demás sentencias (asignación, expresión en una línea, return ...) sí.

Un método se definirá en el interior de la clase mediante una palabra de tipo o la palabra reservada *void*, seguida del nombre del método y dos paréntesis y dos llaves. En el interior de los paréntesis se escribirán, separados por comas, los argumentos del método dados por el tipo de la variable y su nombre. Estos argumentos serán variables que tendrán como ámbito el propio método. Dentro de las llaves irá el código que se ejecutará cuando se llame al método. El método deberá contener una instrucción return, esto es, la palabra reservada *return* seguida de una variable o constante del tipo que se especificó al inicio de su declaración. Si este tipo era *void*, solo debe aparecer return y punto y coma.

Se podrán realizar llamadas a métodos de una clase mediante un punto tras el nombre de un objeto y escribiendo el nombre (y dos paréntesis) de un método perteneciente a la clase correspondiente al tipo del objeto, todo sin espacios de por medio. Si se omite el nombre de un objeto y el punto, es decir, solo escribiendo el nombre del método y dos paréntesis, el método deberá estar en la propia clase. Dentro de los paréntesis irán, separados por comas, las constantes o los nombres de las variables que se pasarán al método, por valor si son tipos básicos y por referencia si son arrays u objetos. Deberán pasarse el número exacto de argumentos y deberán tener el mismo tipo que aparece en la declaración del método.

Nótese que en la inicialización de un array o un objeto a un valor, se realiza una instrucción de reserva de memoria mediante la palabra reservada *new*.

Para acceder a la posición de un array se escribirá el nombre de la variable de tipo array seguida de dos corchetes y en su interior un entero no negativo, bien como literal o como variable (en cuyo caso se comprobará solamente que es de tipo entero).

Las expresiones estarán formadas por constantes de los tipos básicos, variables de cualquier tipo (con y sin subíndices), llamadas a métodos y los operadores anteriormente explicados.

Ejemplos de instrucciones:

```
for(int i=0; i<10; i++) {
     ...
}

// Esto es un comentario
/* Esto también es un comentario */

if (true) {...} else {...}

funcion(); //método de la propia clase
objeto.metodo(); //método de la clase del objeto
```

• Gestión de errores

Los errores de compilación mostarán el tipo de error y la fila y columna del comienzo del error y la fila y columna del final del error (que será donde la recuperación de errores haya podido seguir). Habrá recuperación de errores saltando al punto y coma o llave (de cierre) más cercano, descartando los paréntesis que pudiese haber abiertos o cerrados por el camino.

• Inicio de ejecución

Deberá haber alguna clase, y solo una, con un método llamado main sin argumentos y de tipo void, ya que ahí comenzará la ejecución del programa.

• Ejemplo

```
class Buscador() {

     Buscador() {}

     boolean buscarLetra(char[] array, char letra, int tam) {
          boolean encontrado = false;
          for(int i=0; i<size && encontrado==false; i++) {
               if(array[i]==letra) {
                    encontrado = true;
               }
          }
          return encontrado;
     }
}

class Main {
     void main() {
          char[] array = getArray();
          int tam = 6;
          char c = ‘e’;
          Buscador buscador = new Buscador();
          boolean esta = buscador.buscarLetra(array,c,tam);
     }

     char[] getArray() {
          char[] array = new char[]{‘P’,’r’,’u’,’e’,’b’,’a’};
          return array;
     }
}
```