Proyecto de Curso
Reduced Java

Fundamentos de Lenguajes Programación
Escuela de Ingeniería de Sistemas y Computación
Universidad del Valle

Profesor Carlos Alberto Ramírez Restrepo
carlos.a.ramirez@correounivalle.edu.co

1. Introducción

El presente proyecto tiene por objeto enfrentar a los estudiantes del curso:

- A la implementación de un lenguaje de programación.
- Al uso de ayudas para la creación de interpretadores y compiladores.
- Al análisis de estructuras sintácticas, de datos y de control de un lenguaje de programación para la optimización de un interpretador o compilador asociado a él.

2. R-Java

R-Java (*Reduced Java*) es un lenguaje de programación pequeño, versátil, hábil e imperativo basado en el lenguaje de programación Java. El propóstio de *R-Java* es combinar las principales características de Java como Lenguaje Imperativo y Orientado a objetos con algunos conceptos y estructuras adicionales. *R-Java* es sintácticamente similar a Java aunque posee algunas modificaciones que garantizan que su gramática sea LL1.

A continuación se describen las principales características de *R-Java*:

1. *R-Java* es un lenguaje que soporta orientación a objetos en el cual es posible definir clases y objetos, así como definir funciones o procedimientos.

2. Un programa en *R-Java* consiste de un conjunto (posiblemente vacío) de declaraciones de clases, de un conjunto de declaraciones de procedimientos y de un conjunto de instrucciones principales o expresiones (programa principal).

3. Los nombres de las variables en *R-Java* corresponden a secuencias de una o mas letras o dígitos empezando con una letra en minúscula.

4. Los nombres de los procedimientos y clases en *R-Java* corresponden a secuencias de una o mas letras o dígitos empezando con una letra en mayúscula.

5. Todas las expresiones en *R-Java* tienen un valor. Sin embargo, existen bloques de códigos llamados instrucciones o sentencias los cuales producen efectos pero no retornan ningún valor.

6. Todas las variables en *R-Java* son de múltiple asignación (celdas).

7. Los tipos básicos en *R-Java* pueden ser cadenas, carácteres, números y booleanos. Además se tiene que *R-Java* soporta listas, estructuras, arreglos y vectores.

8. *R-Java* posee las expresiones condicionales **if elseif else** y **swith case** de Java.

9. *R-Java* incorpora los ciclos **for**, **while** y **do while**.

10. Las clases en *R-Java* deben poseer un método constructor del mismo nombre de la clase (similar a Java). Este método es ejecutado en el momento de la creación de un objeto. En caso de no especificarse este método debe desplegarse un error.

11. *R-Java* posee arreglos como estructuras del lenguaje y posee mecanismos para la creación de arreglos, así como el acceso y modificación de valores en ellos. Tambien debe definirse la primitiva `length` que determina el tamaño de un arreglo.

12. *R-Java* posee vectores como tipo de dato del lenguaje y posee mecanismos para la creación de vectores, así como el acceso y modificación de valores en ellos. Además, incorpora mecanismos que permiten la extensión, reducción y consulta del tamaño de un vector.

13. *R-Java* posee manejo de estructuras. *R-Java* incorpora mecanismos para definición de plantillas e instancias de estructuras. Igualmente, *R-Java* implementa la primitiva `instanceOf` que verifica si un valor dado corresponde a una instancia de una estructura en particular.

    Además de los elementos mencionados para el manejo de estructuras. *R-Java* proporciona el operador **cases** el cual permite ejecutar un conjunto de instrucciones a partir de un valor (que puede ser un número o una estructura).

    vectores como tipo de dato del lenguaje y posee mecanismos para la creación de vectores, así como el acceso y modificación de valores en ellos. Además, incorpora mecanismos que permiten la extensión, reducción y consulta del tamaño de un vector.

14. *R-Java* soporta los operadores `++`, `--`, `+=`, `-=`, `*=`, `/=` y el operador de asignación `=`.

15. *R-Java* incorpora las operaciones primitivas de valores enteros suma, resta, multiplicación, división, módulo, inverso, potencia, `==`, máximo, mínimo, `>`, `<`, `<=` y `>=`. Estas operaciones pueden tener un número arbitrario de parámetros en los casos donde tenga sentido o uno o dos parámetros donde corresponda.

16. *R-Java* incorpora las operaciones primitivas de valores booleanos `and`, `or` y `not`.

17. *R-Java* incorpora las primitivas de listas `cons`, `car`, `cdr`, `null?`, `empty`, `list`, `length` y `append`.

18. *R-Java* incorpora la primitiva para cadenas `append`.

```
main()
{
  (+ 4 5 6 7)
}
```

Figura 1: Un programa sencillo.

19. *R-Java* incorpora la expresión **local** y la instrucción **set**.

20. Los métodos de los objetos en *R-Java* retornan el valor de la última expresión en su definición.

21. Los objetos en *R-Java* se pueden clonar.

22. *R-Java* es un lenguaje tipado fuerte y estático que soporta inferencia de tipos.

3. Ejemplos

En esta sección se muestran algunos ejemplos de programas escritos en *R-Java*. Los ejemplos son mostrados en las Figuras 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 y 11.

En la Figura 1, se muestra un programa sencillo donde se suman varios valores enteros. Este programa tiene como valor 22.

En la Figura 2, se muestra un ejemplo del uso de las primitivas de comparación de valores enteros y del uso de condicionales. Este programa debe retornar 9.

En la Figura 3, se muestra un ejemplo de la creación de procedimientos, del uso de condicionales y del uso de las primitivas `max` y `min`. Este programa debe retornar 0.

En la Figura 4, se muestra un programa donde se evidencia que todas las variables en *R-Java* son de

```
main()
{
  local
    x y
  in
   set x = 10;
   set y = 5,

   if((&& (<= x 6) (== y 5)))
   {
      ++x;
   }
   elseif((&& (> x 6) (<= y 4)))
   {
      (- x y);
   }
   else
   {
      --x;
   };
  end
}
```

Figura 2: Un programa con condicionales.

múltiple asignación. El valor de este programa es 10.

En la Figura 5, se muestra un programa donde se manipulan listas. El valor de este programa es la lista `(6)`.

En la Figura 6, se ilustra un programa en el cual se define un procedimiento que utiliza un **switch case**. El valor de este programa es 14.

En la Figura 7, se muestra un programa donde se hace uso de la estructura de ciclos **for** para definir un procedimiento que calcula la sumatoria de un número. En tanto que en la figura 8, se muestra el uso de la forma **while** para definir un procedimiento que calcula el factorial. Los valores de estos programas son respectivamente 55 y 120.

En la Figura 9, se muestra la forma en que se

```
proc(F x y z)
{
   if(x)
   {
     (max y z);
   }
   else
   {
     (min y z);
   }
}

main()
{
  local
    m
  in
    set m = 0,
    (F true m -4);
  end
}
```

Figura 3: Un programa con procedimientos.

```
main()
{
  local
    x = 0;
  in
    set x = 0;
    set x += 10,
    x;
  end
}
```

Figura 4: Un programa con asignación.

```
main()
{
  local
    x y
  in
    set x = (cons 5 (cons 6 empty));
    set y = (cdr x),
    y;
  end
}
```

Figura 5: Un programa con listas.

```
proc(F x y)
{
  switch(x):
  {
    case 1:
      (+ y 1);
      break;

    case 2:
      (+ y 2);
      break;

    case 3:
      (+ y 3);
      break;

    default:
      y;
  }
}

main()
{
  local
    m
  in
    set m = 5,
    (* (F 2 m) 2);
  end
}
```

Figura 6: Un programa con **switch case**.

```
proc(Sum x)
{
  local
     sum
  in
    for(y=1; (<= y x); ++y)
    {
      set sum += y;
    },
    sum;
  end
}

main()
{
  (Sum 10)
}
```

Figura 7: Un programa con **for**.

```
proc(Fact x)
{
  local
     res y
  in
     set y=1;
     set res = 1;
     while((<= y x))
     {
       set res *= y;
       ++y;
     },

     res;
  end;
}

main()
{
  (Fact 5)
}
```

Figura 8: Un programa con **while**.

definen y utilizan arreglos en *R-Java*. En este programa se define un procedimiento que calcula la sumatoria de los valores almacenados en un arreglo, luego se define un arreglo `a` de 5 posiciones que inicialmente tienen como valor 3. Posteriormente se modifican los valores en las posiciones 0 y 1 del arreglo y finalmente se obtiene el valor de la sumatoria de los elementos de `a`. Este programa arroja 16.

En la Figura 10 se muestra un programa donde se definen y utilizan estructuras. Inicialmente se define una plantilla de estructura llamada `str`. Luego, se crea una instancia de dicha estructura y se almacena en la variable `e`. Finalmente, se accede al campo `c1` y el programa retorna 1.

En la Figura 11 y 12 se muestran programas donde se definen clases y se crean objetos de dichas clases. En el programa de la figura 10 se crea una clase `c1` que almacena dos valores que pueden ser actualizados y obtenidos a través de sus métodos. El valor de este programa es la lista `((3 -3) (5 -5))`. Mientras que en el programa de la Figura 12 se define un arbol mediante clases. Se define en la clase arbol un método para obtener la sumatoria de los valores en los nodos de un arbol, Finalmente se crea un objeto de esta clase y se calcula el valor de la sumatoria de sus nodos. El valor de este programa es 12.

4. Trabajo a realizar

Cada grupo debe entregar lo siguiente:

a) La definición de la gramática y la especificación léxica, sintáctica y semántica del lenguaje *R-Java*.

b) La implementación del lenguaje *R-Java*.

c) Opcionalmente, cada grupo puede implementar características adicionales útiles.

```
proc(Sumarray arr)
{
   local
      sum tam
   in
      set tam = (length arr);
      for(x=0; (<= x tam); ++x)
      {
        set sum += [arr x];
      },

      sum;
   end
}

main()
{
  local
    a
  in
    set a = (newarray 5 3);
    (setarray a 0 1);
    (setarray a 1 6),
    (Sumarray a);
  end
}
```

Figura 9: Un programa con arreglos.

```
main()
{
   local
      str e
   in
      set str=(define-struct c1 c2);
      set e=(make-struct str 1 2),
      .e.c1;
   end
}
```

Figura 10: Un programa con estructuras.

```
class C1 extends object
{
  i,j;

  method C1(x)
  {
    set i = x;
    set j = -(0,x);
  }

  method Countup(d)
  {
    set i = +(i,d);
    set j = -(j,d);
  }

  method Getstate()
  {
    (list i j);
  }
}

main()
{
  local
    t1 t2 o1
  in
    set o1 = new C1(3);

    set t1 = class.o1.Getstate();
    class.o1.Countup(2);
    set t2 = class.o1.Getstate(),
    (list t1 t2);
  end
}
```

Figura 11: Un programa con clases.

```
class Interior_node extends object
{
  left, right;

  method Interior_node(l, r)
  {
    set left = l;
    set right = r;
  }

  method Sum()
  {
    (+ class.left.Sum() class.right.Sum());
  }
}

class Leaf_node extends object
{
  value;

  method Leaf_node(v)
  {
    set value = v;
  }

  method Sum()
  {
    value;
  }
}

main()
{
  local
    o1
  in
    set o1 = new Interior_node(
                new Interior_node(
                   new Leaf_node(3)
                   new Leaf_node(4))
                new Leaf_node(5)),
    class.o1.Sum();
  end
}
```

Figura 12: Un programa con clases II.

Aclaraciones

- ☞ El proyecto se debe desarrollar en grupos de máximo tres (3) alumnos.
- ☞ En todo momento se debe tener en cuenta las restricciones y condiciones impuestas.
- ☞ Existe libertad para hacer cambios en la gramática pero dichos cambios deben ser claramente sustentados.
- ☞ Recuerdo que su equipo de trabajo es una unidad, lo cual significa que debe haber comunicación entre los miembros, donde cada cual puede trabajar en aspectos diferentes del proyecto, pero debe conocer (al menos someramente) el trabajo de los demás.
- ☞ No se debe compartir información específica de las implementaciones, ni tampoco debe haber reuniones entre varios grupos.
- ☞ Cualquier indicio de fraude o copia, asignará la nota de CERO (0.0) a todos los miembros del equipo. Tener en cuenta el punto anterior, es decir, si varios equipos hacen reuniones, puede dar un indicio de copia.
- ☞ El equipo de trabajo debe realizar el trabajo por si mismo. No se recomienda DE NINGUNA FORMA que se busque ayuda externa para la realización de la implementación del proyecto. Solamente busque ayuda del profesor.
- ☞ Utilice únicamente las herramientas que usted maneje y entienda completamente, recuerde que se debe sustentar el trabajo.
- ☞ Para la calificación se tendrá en cuenta dos aspectos, la especificación e implementación del lenguaje y la sustentación. Para la sustentación cada estudiante debe tener la capacidad de explicar, entender, modificar y añadir pequeñas características al lenguaje.

- ☞ Se debe entregar el código fuente de la implementación del lenguaje. Así mismo se debe presentar un informe (pdf) donde se especifique las decisiones tomadas para definir el lenguaje. Se debe presentar la descripción de la gramática, la definición de las estructuras y tipos de datos definidos y utilizados y las técnicas empleadas.

- ☞ Se debe comentar en el código fuente cada función definida especificando el tipo de dato de entrada y salida, así como una breve descripción de la utilidad de dicha función.

- ☞ El proyecto tendrá una entrega parcial y una entrega final. A continuación se describe las especificaciones de cada una de las entregas:

  a) **Entrega Parcial**: En esta primera entrega se debe presentar la definición de la especificación léxica y la gramática de *R-Java* en Scheme. Además se debe presentar la definición de las estructuras y tipos de datos que usará el interpretador. Esto incluye la definición del tipo de dato para ambientes, variables, listas, estructuras, vectores, arreglos, clases, etc. Igualmente se debe presentar la definición de las operaciones que usan estos tipos de datos. Esta entrega tiene como fecha máxima el día **16 de Mayo a las 10:00 pm**. Se debe subir al campus un archivo que siga la convención *Apellido1Apellido2Apellido3-Entrega1.zip*.

  b) **Entrega Final**: La entrega final del proyecto corresponde a la definición e implementación del interpretador y el informe. Esta entrega se debe subir al campus a más tardar el día **8 de Junio a las 10:00 pm**. Se debe subir al campus un archivo adjunto que siga la convención *Apellido1Apellido2Apellido3 ProyectoFLP14.zip*.

- ☞ La sustentación del proyecto se llevara a cabo el día **9 de Junio** entre las 9am y 4pm.

- ☞ Las fechas de entrega y sustentación del proyecto solo se modificarán teniendo en cuenta la disponibilidad de todos los estudiantes y de los espacios en la universidad.

- ☞ Recuerde, en caso de dudas y aclaraciones puede preguntar al inicio de las clases, asistir al horario de consulta (lunes y jueves de 2pm a 4pm) o enviar un correo con su consulta al profesor.