Fundamentos de Lenguajes de Programación: Proyecto **OBLIQ**

Escuela de Ingeniería de Sistemas y Computación
Facultad de Ingeniería

1. Introducción

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

- a la comprensión de todos los conceptos vistos en clase.
- A la implementación de un interpretador de un lenguaje de programación.
- Al análisis de estructuras sintácticas, de datos y de control de un lenguaje de programación para la implementación de un interpretador o compilador asociado a él.

2. El Lenguaje Obliq

Obliq [1] es un lenguaje de programación interpretado que soporta programación orientada a objetos y tiene una estructuras de paso de parámetros por valor. La sintaxis y semántica de Obliq es descrita a continuación:

2.1. Estructuras Sintácticas

- **Identificadores:** Son secuencias de caracteres alfanuméricos que comienzan con una letra.
- **Definiciones:** Comienzan con `let`, `let rec` o `var`, seguido de una lista de ligaduras de variables a expresiones, separados por comas.
- **Expresiones:** En Obliq, casi todas las estructuras sintácticas son una *expresión*, y todas las expresiones producen un valor. Las expresiones pueden ser clasificadas en: *Identificadores*, *Datos*, *Constructores*, *Operaciones*.

2.2. Estructuras de Datos

- **Constantes:**

`true`, `false`	booleanos.
`0`, `1`, `~1`	enteros.
`' a '`	caracteres.
`" abc "`	cadenas.
`ok`	

- **Operadores:**

```
bool:   not and or
int:    + - * / % > < >= <= is
text:   &
```

- **Objetos:** Los objetos son una colección de *campos* $\texttt{x}_\texttt{i} \texttt{=>} \texttt{a}_\texttt{i}$, donde $\texttt{x}_\texttt{i}$ es un *nombre de campo*, y $\texttt{a}_\texttt{i}$ es una expresión.

  $\texttt{object}\{\texttt{x}_1 \texttt{=>} \texttt{a}_1, \ldots, \texttt{x}_\texttt{n} \texttt{=>} \texttt{a}_\texttt{n}\}$ para $\texttt{n} \geq 0$.

  Para seleccionar un campo de un objeto, se escribe `get a.x`, donde `a` es el objeto, y `x` es el campo.

  La invocación de un método se realiza con `send` $\texttt{a.x}(\texttt{b}_1, \ldots, \texttt{b}_\texttt{n})$, siendo `a` el objeto, `x` el método y $\texttt{b}_1, \ldots, \texttt{b}_\texttt{n}$ los argumentos.

  La clonación de un objeto se hace con $\texttt{clone}(\texttt{a}_1, \ldots, \texttt{a}_\texttt{n})$ (herencia múltiple).

2.3. Estructuras de Control

- **Definiciones:**

  $\texttt{var } \texttt{x}_1 = \texttt{a}_1, \ldots, \texttt{x}_\texttt{n} = \texttt{a}_\texttt{n} \texttt{ b end}$
  $\texttt{let } \texttt{x}_1 = \texttt{a}_1, \ldots, \texttt{x}_\texttt{n} = \texttt{a}_\texttt{n} \texttt{ b end}$
  $\texttt{let rec } \texttt{x}_1 = \texttt{p}_1, \ldots, \texttt{x}_\texttt{n} = \texttt{p}_\texttt{n} \texttt{ b end}$

  Una definición `var` introduce una colección de variables actualizables y sus valores iniciales. Una definición `let` introduce una colección de identificadores no actualizables y sus valores. Una definicón `let rec` introduce una conlección de procedimientos recursivos.

- **Asignación:**

  $\texttt{x := a}$,
  donde `x` es una variable y `a` es una expresión. Ejemplo:
  `x := +(x 1)`

- **Secuenciación:**

  $\texttt{a}_1; \ldots; \texttt{a}_\texttt{n}$

  Un conjunto de definiciones y términos, puede ser evaluado secuencialmente, separándolos con "punto y coma". Ejemplo:

  `var x=3; set x:=+(x 1); x`
  devolverá 4.

- **Procedimientos y Métodos:**

  $\texttt{proc}(\texttt{x}_1, \ldots, \texttt{x}_\texttt{n})\texttt{ b end}$
  $\texttt{meth}(\texttt{s}, \texttt{x}_1, \ldots, \texttt{x}_\texttt{n})\texttt{ b end}$
  en donde **b** es una expresión, $\texttt{x}_\texttt{i}$ son los argumentos y `s` es el self.

- **Condicionales:**

  $\texttt{if } \texttt{a}_1 \texttt{ then } \texttt{a}_2 \texttt{ elsif } \texttt{a}_3 \texttt{ then } \texttt{a}_4 \ldots \texttt{ else } \texttt{a}_\texttt{n} \texttt{ end}$

- **Iteraciones:**

  $\texttt{for x} = \texttt{a}_1 \texttt{ to } \texttt{a}_2 \texttt{ do } \texttt{a}_3 \texttt{ end}$
  en donde $\texttt{a}_1$ y $\texttt{a}_2$ son expresiones que evalúan a un número.

3. Definición Formal del Proyecto

El objetivo del proyecto es crear un interpretador de Obliq que ejecute programas como los que encuentra en los anexos. La BNF del lenguaje Obliq también la encuentra en los anexos.

4. Técnicas utilizadas

El proyecto es en grupos de 3 personas, quienes deben desarrollar el interpretador con las herramientas que se han visto en clase, es decir, con la metodología para la realización de un interpretador y el uso de SLLGEN. Todas las funciones deben estar documentadas, la documentación es un punto importante del proyecto.

5. Implementación

Se debe entregar un solo archivo que contiene el interpretador y el informe. En la entrega deben incluir un archivo con el nombre **proyecto-flp.scm** que debe contener todos los procedimientos necesarios para el buen funcionamiento de un interpretador de Obliq. El interpretador se probará con los ejemplos dados y otros derivados de ellos.

6. Informe y Conclusiones

El informe debe contener: la sintáxis del lenguaje, el comportamiento del interpretador para cada una de las expresiones y las descripciones de los procedimientos y datos auxiliares utilizados para el buen funcionamiento del interpretador. También debe existir una sección de conclusiones, en ella se espera que usted analice los resultados obtenidos y **justifique** claramente sus afirmaciones.

Referencias

[1] Luca Cardelli. *A Language with Distributed Scope.* POPL '95: Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages. páginas 286-297. 1995.

7. Anexos

7.1. Ejemplos

7.1.1. Factorial

```
let rec
  Fact(n)= if is(n 0) then 1 else *(n apply Fact(-(n 1))) end

  apply Fact(5)
end
```

7.1.2. Numeros Primos

```
let primo =
  object { m =>
               meth(s, n)
                 n;
                 let s0 = clone(s);
                     update s.m :=
                                  meth(s1,n1)
                                    if is(%(n1 n) 0) then
                                      ok
                                    else
                                      send s0.m(n1)
                                    end
                                  end
               end
         };
  (* nuestra los primos < 100 *)
  for i = 2 to 100 do send sieve.m(i) end
end
```

7.1.3. Calculadora

```
let calc =
  object{
          arg => 0,
          acc => 0,

          enter =>       (* entra un nuevo argumento *)
                  meth(s, n)
                    update s.arg := n;
                    s
                  end,

          add =>         (* la suma *)
                meth(s)
                  update s.acc := send s.equals;
                  update s.equals := meth(s) +(get s.acc get s.arg) end;
                   s
                end,

          sub =>         (* la resta *)
```

```
                meth(s)
                  update s.acc := send s.equals;
                  update s.equals := meth(s) -(get s.acc get s.arg) end;
                  s
                end,

          equals =>      (* el resultado *)
                   meth(s)
                     get s.arg end,

          reset =>       (* inicializar *)
                  meth(s)
                    update s.arg := 0;
                    update s.acc := 0;
                    update s.equals := meth (s) get s.arg end;
                    s
                  end
      };

  send calc.reset(); send calc.enter(3); send calc.equals(); (* 3 *)

  send calc.reset(); send calc.enter(3); send calc.sub();
  send calc.enter(2); send calc.equals(); (* 1 *)

  send calc.reset(); send calc.enter(3); send calc.add();
  send calc.equals(); (* 6 *)

  send calc.reset(); send calc.enter(3); send calc.add();
  send calc.add(); send calc.equals; (* 9 *)

end
```