Lenguajes de Programación.
Ejercicios F1WAE
Profesor: Paul Leger
Ayudante: José Carrillo

1. ¿Qué tipo de valor usa siempre el intérprete F1WAE?
    a. Número
    b. Función
    c. Void
    d. Todas las anteriores

2. ¿Qué valores del lenguaje posee F1WAE?
    a. Números
    b. Funciones
    c. Strings
    d. A y B
    e. Todas las anteriores
3. En el caso de que exista la división a/b, ¿Dónde se gatillara el error en el caso de que b sea igual a 0?
    a. BNF
    b. Parse
    c. Intérprete
    d. AST
    e. Sustitución
4. ¿Qué hace el siguiente código?

```
(define (parse program)
  (cond
    [(number? program) (num program)]
    [(symbol? program) (id program)]
    [(list? program)
     (case (first program)
       [(+) (add (parse(second program))
                 (parse(third program)))]
       [(-) (sub (parse(second program))
                 (parse(third program)))]

       [(with) (with (first (second program))
                     (parse (second (second program)))
                     (parse (third program)))]

       [else (app (first program) (parse (second program)))])]))
```

    a. Define los nodos que puede tener el AST
    b. Construye el AST para que sea interpretado
    c. Transforma la sintaxis abstracta a sintaxis concreta
    d. Define la sintaxis concreta del lenguaje

5. ¿Qué hace el siguiente fragmento de código?

```
(define-type F1WAE
  [num (n number?)]
  [add (l F1WAE?) (r F1WAE?)]
  [sub (l F1WAE?) (r F1WAE?)]
  [with (name symbol?) (value F1WAE?) (body F1WAE?)]
  [id (name symbol?)]
  [app (fun-name symbol?) (arg F1WAE?)])
```

   a. Define los nodos que puede tener el AST
   b. Construye el AST para que sea interpretado
   c. Transforma la sintaxis abstracta a sintaxis concreta
   d. Define la sintaxis concreta del lenguaje

6. ¿Qué hace el siguiente fragmento de código?

```
(define (subst expr sub-id val)
  (type-case F1WAE expr
             [num (n) expr]
             [add (l r) (add (subst l sub-id val)
                             (subst r sub-id val))]
             [sub (l r) (sub (subst l sub-id val)
                             (subst r sub-id val))]
             [with (bound-id named-expr bound-body)
                   (if (symbol=? bound-id sub-id)
                       (with bound-id
                             (subst named-expr sub-id val)
                             bound-body)
                       (with bound-id
                             (subst named-expr sub-id val)
                             (subst bound-body sub-id val)))]
             [id (v) (if (symbol=? v sub-id) val expr)]
             [app (fun-name arg-expr) (app fun-name (subst arg-expr sub-id val))]
             ))
```

   a. Reemplaza todas las ocurrencias de una variable en el AST por su respectivo valor “val”
   b. Realiza el proceso de sustitución de variables en el parser
   c. Realiza el proceso de sustitución de variables en el intérprete
   d. Sustituye automáticamente todas las variables que aparecen en el código por el valor “1”, sin tener en cuenta la variable o su valor específico.

Utiliza el siguiente código para responder las preguntas 7, 8 y 9

```
(define-type F1WAE
  [num (n number?)]
  [add (l F1WAE?) (r F1WAE?)]
  [sub (l F1WAE?) (r F1WAE?)]
  [with (name symbol?) (value F1WAE?) (body F1WAE?)]
  [id (name symbol?)]
  [app (fun-name symbol?) (arg F1WAE?)])
```

```
(define (parse program)
  (cond
    [(number? program) (num program)]
    [(symbol? program) (id program)]
     (case (first program)
       [(+) (add (parse(second program))
                 (parse(third program)))]
       [(-) (sub (parse(second program))
                 (parse(third program)))]

       [(with) (with (first (second program))
                     (parse (second (second program)))
                     (parse (third program)))]
       [else (app (first program) (parse (second program)))])]))
```

```
(define (interp expr fun-defs)
  (type-case F1WAE expr
             [num (n) n]
             [add (l r) (+ (interp l fun-defs) (interp r fun-defs))]
             [sub (l r) (- (interp l fun-defs) (interp r fun-defs))]
             [with (bound-id named-expr bound-body)
                   (interp (subst bound-body
                                  bound-id
                                  (num (interp named-expr fun-defs)))
                           fun-defs)]
             [id (v) (error 'interp "free identifier")]
             [app (fun-name arg-expr)
                  (let ([the-fun-def (lookup-fundef fun-name fun-defs)])
                    (interp (subst (fundef-body the-fun-def )
                                   (fundef-arg-name the-fun-def )
                                   (num (interp arg-expr fun-defs)))
                            fun-defs))]))
```

```
(define (subst expr sub-id val)
  (type-case F1WAE expr
             [num (n) expr]
             [add (l r) (add (subst l sub-id val)
                             (subst r sub-id val))]
             [sub (l r) (sub (subst l sub-id val)
                             (subst r sub-id val))]
             [with (bound-id named-expr bound-body)
                   (if (symbol=? bound-id sub-id)
                       (with bound-id
                             (subst named-expr sub-id val)
                             bound-body)
                       (with bound-id
                             (subst named-expr sub-id val)
                             (subst bound-body sub-id val)))]
             [id (v) (if (symbol=? v sub-id) val expr)]
             ;;extension: posible substituciones dentro la funciones
             [app (fun-name arg-expr) (app fun-name (subst arg-expr sub-id val))]
             ))
```

7. Modifica el código anterior, de manera que, cuando se encuentre un identificador libre, el intérprete retorne el valor 5.

8. Modifica el código anterior para que, al encontrar cualquier ocurrencia de ‘z’, esta sea reemplazada automáticamente por el valor 100.

9. Genere para el código anterior el Azúcar Sintáctico ****, el cual retornará el cuádruple de un número.

10. Imaginemos que a la hora de realizar la interpretación del AST en el lenguaje F1WAE, encontramos el nodo ‘id x’. ¿Qué significa esto?
    a. Que aún no se realiza la sustitución de esta variable.
    b. Que nunca se le asignó un valor a la variable.
    c. Que se está interpretando un nodo with.
    d. Que se posee un scope dinámico.

11. Defina los siguientes conceptos
    a. Azúcar sintáctico

    b. Intérprete

    c. BNF

    d. Bound Instance

Utiliza el siguiente fragmento de código para las responder las preguntas 12 y 13

```
(define (subst expr sub-id val)
  (type-case F1WAE expr
             [num (n) expr]
             [add (l r) (add (subst l sub-id val)
                             (subst r sub-id val))]
             [sub (l r) (sub (subst l sub-id val)
                             (subst r sub-id val))]

             [with (bound-id named-expr bound-body)

                   (if (symbol=? bound-id sub-id)

                       (with bound-id
                             (subst named-expr sub-id val)
                             bound-body)
                       (with bound-id
                             (subst named-expr sub-id val)
                             (subst bound-body sub-id val)))]
             [id (v) (num 1)]
             [app (fun-name arg-expr) (app fun-name (subst arg-expr sub-id val))]
             ))
```

12. ¿Qué retorna la expresión {with {x 5} {+ x {with {x 3} 10}}} al interpretarla?
    a. 10
    b. 5
    c. 3
    d. 11
    e. 15
    f. 2

13. ¿Qué retorna la expresión {with {x 5} {with {x y} x}} al interpretarla?

a. ‘x
b. ‘y
c. 1
d. 5
e. 10
f. “free identifier”
g. void
h. Otro tipo de error

14. Modifica el siguiente fragmento de código, de tal manera que, al no encontrar una función, se retorne la función cuadrado. (Asuma que la expresión “mul” existe)

```
(define (lookup-fundef fun-name fundefs)
  (cond
    [(empty? fundefs) (error fun-name "function not found")]
    [else (if (symbol=? fun-name (fundef-fun-name (first fundefs)))
              (lookup-fundef fun-name (rest fundefs)))]))
```