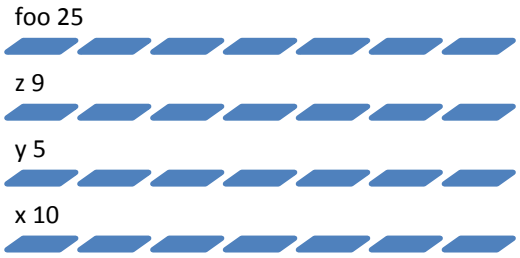
Problema I:

- ***Provee un esquema para un programa que ilustre la no-linealidad de la implementación de ambientes basada en un stack. Explica brevemente porque su ejecución en tiempo no es lineal con respecto al tamaño de su entrada.***

```
{with {x 10}
   {with {y 5}
      {with {z 9}
         {with {foo {fun {w} {+ x {+ y {+ z w}}}}}
            {foo 1}}}}}
```

Este ejemplo va a realizar un subs por cada with y cada aplicación que tengamos. Este programa tiene tamaño 4 y en cada línea se realiza una sustitución, por lo que no puede ser lineal, siendo éste $O(n^2)$.

La pila con cada subs quedaría de la siguiente manera:



- ***Describe una estructura de datos para un ambiente que un intérprete de FWAE pueda usar para mejorar su complejidad***

Utilizar Tablas Hash bien diseñadas, usando el manejo de colisiones ayudará a mejorar la complejidad a tiempo constante, sin importar el número de elementos que tengamos.

- ***Muestra como usaría el intérprete esta nueva estructura de datos.***

Como sabemos, las tablas Hash necesitan una llave y un elemento, en este caso la llave sería el identificador del with, y el elemento sería el valor de ese identificador.

```
(define (interp expr)
   (type-case FWAE expr
 [num (n) expr]
 [add (l r) (add-numbers (interp l) (interp r))]
 [with (llave  elemento bound-body)
         (interp (subst bound-body
                              llave
                              (interp  elemento)))]
[id (v) (error 'interp "free identifier")]
[fun (llave  bound-body)
        expr]
[app (fun-expr arg-expr)
       (local ([define fun-val (interp fun-expr)])
          (interp (subst (fun-body fun-val)
                              (fun-param fun-val)
                              (interp arg-expr)))))] ))
```

Los cambios que realizaremos a la función interp serán:
-Sustituir el bound-id por la llave (que como se explicó arriba sera el identificador del with)
-Sustituir el named-expr por el elemento (que como también se explico arriba será el valor del identificador)

- ***Indica cual es la nueva complejidad del interprete (análisis del peor caso) y de forma informal pero rigurosa pruébalo.***

La complejidad en el peor caso sería O(n), esto es, constante. Ya que la búsqueda en una tabla hash para subs dependería de su tamaño, en este (el peor caso), su longitud sería *n,* por lo que la búsqueda se realizaría de manera secuencial.

Problema II:

Contraejemplo:

```
{with {x 4}
   {with {y 2}
      {with {f {fun {z} {+ z {+ x  y}}}}
         {with {x 5}
            {with {y 7}
               {f 10}}}}}}
```

Ben esta afirmando que conservando el valor más viejo de ‘x’ se puede seguir usando alcance dinámico, pero olvido que hay programas que pueden tener otras variables además de ‘x’ que tomen diferente valor dependiendo el tipo de alcance que usemos.

En el contraejemplo dado arriba podemos notar que ‘y’ va a tomar diferente valor si usamos alcance dinámico a si usamos alcance estático, por lo que aun conservando el valor más viejo de ‘x’ el resultado será diferente de acuerdo al método propuesto por Ben.

Evaluemos el contraejemplo:

Alcance estático:

```
{with {x 4}
   {with {y 2}
      {with {f {fun {10} {+ 10 {+ 4  2}}}}
         {with {x 5}
            {with {y 7}
              {f 10}}}}}}
```

Resultado: 16

Alcance dinámico:

```
{with {x 4}
   {with {y 2}
      {with {f {fun {10} {+ 10 {+ 5  7}}}}
         {with {x 5}
            {with {y 7}
               {f 10}}}}}}
```

Resultado: 22

Ben:

```
{with {x 4}
   {with {y 2}
      {with {f {fun {10} {+ 10 {+ 4  7}}}}
         {with {x 5}
            {with {y 7}
               {f 10}}}}}}
```

Resultado: 21

Al realizar la evaluación del contraejemplo, nuevamente se comprueba que lo que Ben dice es erróneo.

Problema III:

Dada la siguiente expresión de FWAE con with multi-parametrico:

```
{with {{x 5} {adder {fun {x} {fun {y} {+ x y}}}} {z 3}}
                                    {with {{y 10} {add5 {adder x}}}
                                          {add5 {with {{x {+ 10 z}} {y {add5 0}}}
                                     {+ {+ y x} z}}}}}
```

- ***Da la forma Bruijn de la expresión anterior***

```
{with {5 {adder {fun {x} {fun {y} {+ <0 1> <0 2>}}}} 3}
                                    {with {10 {add5 {adder <0 0>}}}
                                          {add5 {with {{<1 0> {+ 10 <1 3>}} {<0 0> {add5 0}}}
                                     {+ {+ <1 0> <2 0>} <2 3>}}}}}
```

- ***Realiza la corrida de esta expresión, es decir escribe explícitamente cada una de las llamadas tanto para subst y interp, escribiendo además los resultados parciales en sintaxis concreta.***

calc = interp

```
> (calc (parser ' (with ((x 5) (adder (fun (x) (fun (y) (+ x y)))) (z 3))
                                    (with ((y 10) (add5 (adder x)))
                                          (add5 (with ((x (+ 10 z)) (y (add5 0)))
                                     (+ (+ y x) z)))))))

= (calc (with (id x) (num 5) (adder (fun (x) (fun (y) (add (id x) y)))) (z 3))
                                    (with ((y 10) (add5 (adder (id x))))
                                          (add5 (with (((id x) (add 10 z)) (y (add5 0)))
                                     (add (add y (id x)) z))))))

= (calc (subst (adder (fun (x) (fun (y) (add (id x) y)))) (z 3))
                                    (with ((y 10) (add5 (adder (id x))))
                                          (add5 (with (((id x) (add 10 z)) (y (add5 0)))
                                     (add (add y (id x)) z)))))
                                      (id x)
                                      (num (calc (num 5)))))
```

```
= (calc (subst (adder (fun (x) (fun (y) (add (id x) y)))) (z 3))
                                        (with ((y 10) (add5 (adder (id x))))
                                                (add5 (with (((id x) (add 10 z)) (y (add5 0)))
                                         (add (add y (id x)) z)))))
                                          (id x)
                                          (num 5)))

= (calc (subst (adder (fun (x) (fun (y) (add (id x) y)))) (z 3))
                                        (calc (parser ' (with ((y 10) (add5 (adder (id x))))
                                                               (add5 (with (((id x) (+ 10 z)) (y (add5 0)))
                                                        (+ (+ y (id x)) z)))))))))

= (calc (subst (adder (fun (x) (fun (y) (add (id x) (id y))))) (z 3))
                                        (calc (with ((num 10) (add5 (adder (id x))))
                                                               (add5 (with (((id x) (+ 10 z)) ((id y) (add5 0)))
                                                        (add (add (id y) (id x)) z)))))))))

= (calc (subst (adder (fun (x) (fun (y) (add (id x) (id y))))) (z 3))
                                        (calc (subst ((num 10) (add5 (adder (id x))))
                                                               (add5 (with (((id x) (+ 10 z)) ((id y) (add5 0)))
                                                        (add (add (id y) (id x)) z)))))))))
                                                         (id y)
                                                          (num (calc (num 10)))))

= (calc (subst (adder (fun (x) (fun (y) (add (id x) (id y))))) (z 3))
                                        (calc (subst ((num 10) (add5 (adder (id x))))
                                                               (add5 (with (((id x) (+ 10 z)) ((id y) (add5 0)))
                                                        (add (add (id y) (id x)) z)))))))))
                                                         (id y)
                                                         (num 10)))))
```