

Examen 1

CI3641 – Lenguajes de Programación I
Enero–Marzo 2022
Estudiante: Gregory Muñoz
Carnet: 16-11313

Constantes:

$X = 3$

$Y = 1$

$Z = 3$

Enlace de github donde se encontrarán los programas:

<https://github.com/mvgregoryj/CI3641-Lenguajes-de-Programacion-I/tree/main/Examen%201>

Respuesta 1:

Lenguaje escogido: Go

a.i) -Tipo de Alcances:

Go tiene un **alcance estático/léxico** usando bloques.

El alcance de un identificador pre-declarado es el bloque universo.

El alcance de un identificador que denota una constante, tipo, variable o función (pero no método) declarada en el nivel superior (fuera de cualquier función) es el bloque paquete.

El alcance del nombre de paquete de un paquete importado es el bloque de archivo del archivo que contiene la declaración de importación.

El alcance de un identificador que denota un receptor de método, parámetro de función o variable de resultado es el cuerpo de la función.

El alcance de un identificador de constante o variable declarado dentro de una función comienza al final del ConstSpec o VarSpec (ShortVarDecl para declaraciones de variables cortas) y termina al final del bloque contenedor más interno.

El alcance de un identificador de tipo declarado dentro de una función comienza en el identificador en el aspecto tipo y termina al final del bloque contenedor más interno.

Un identificador declarado en un bloque puede volver a declararse en un bloque interior. Si bien el identificador de la declaración interna está en el ámbito de aplicación, indica la entidad declarada por la declaración interna.

-Tipo de Asociaciones:

No hay información sobre el tipo de asociaciones que posee.

a.ii) -Tipos de módulos (si tiene):

-Diferentes formas de importar y exportar nombres

Al importar más de un paquete, se puede crear un bloque.

Forma 1:

bloque de importación de ejemplo antes de la aplicación de formato:

```
import (  
    "fmt"  
    "os"  
    "github.com/digital/ocean/godo"  
    "github.com/sammy/foo"  
    "math/rand"  
    "github.com/sammy/bar"  
)
```

Forma 2:

```
package main
```

```
import (  
    f "fmt"  
    "math/rand"  
)
```

```
func main() {
```

```
    for i := 0; i < 10; i++ {
```

```
        f.Printf("%d) %d\n", i, rand.Intn(25))
```

```
    }
```

```
}
```

Forma 2:

a.iii) -¿Ofrece la posibilidad de crear alises?
No.

-¿Ofrece la posibilidad de crear sobrecarga?

No, Go no tiene funciones sobrecargadas, la función más útil de sobrecarga es la de llamar a una función con argumentos opcionales e inferir valores predeterminados para los omitidos, se puede simular usando una función variable.

-¿Ofrece la posibilidad de crear polimorfismo?

Sí, Golang es un lenguaje orientado a objetos ligero y admite polimorfismo solo a través de interfaces.

- Dar ejemplos de alises, sobrecarga, polimorfismo

a.iv) - Qué herramientas ofrece a potenciales desarrolladores, como:
compiladores, intérpretes, debuggers, profilers, frameworks, etc.

debuggers: Delve

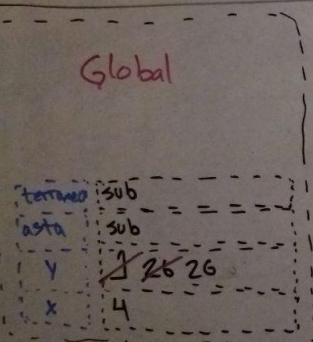
Frameworks: Gin · Beego · Iris · Echo · Fiber

PREGUNTA 2: (a) Alcance estático y asociación profunda

```

int x=4 , y=1;
sub asta (int x){
  y:=2+x;
}
sub terraneo (int y, sub ida, sub urbio){
  if ( y < 8 ){
    sub asta (int x){
      x:=y-2;
      terraneo( y+8 , asta, ida);
    }
  } else if ( y < 16 ){
    terraneo( y+8 , urbio, asta);
  } else {
    int x= 4;
    ida(x+y);
    urbio(x+y);
  }
  Print(X, y)
}
terraneo(x, asta, asta);
Print(x, y)

```



Asociación Profunda y alcance Estático

Imprime:

4	20
4	12
4	4
4	26

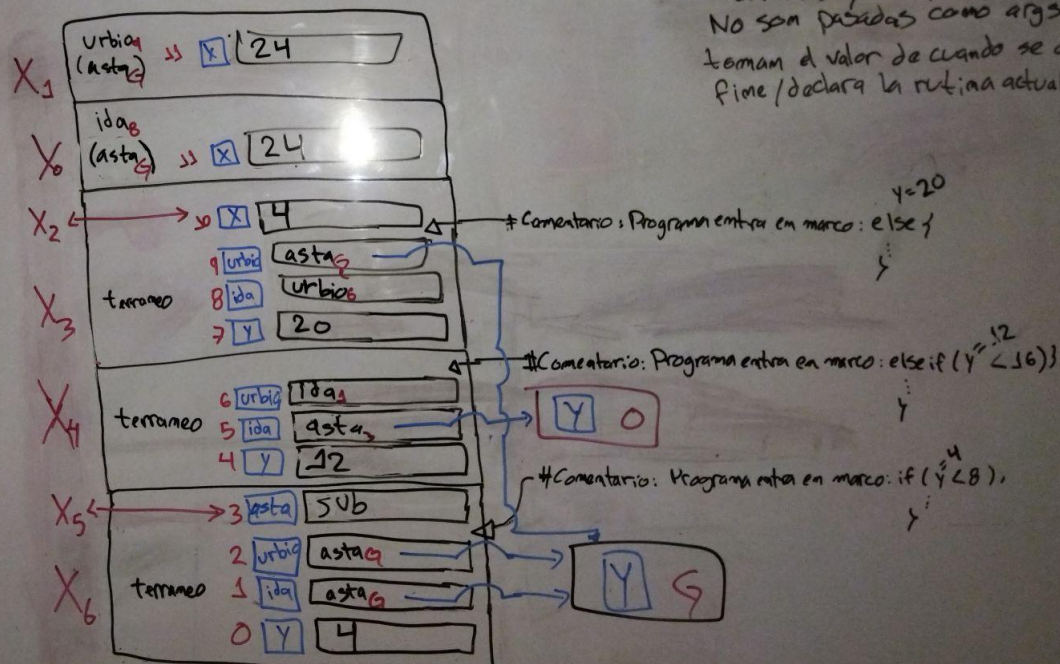
X=3

Y=1

Z=3

Asociación Profunda:
Se crea la closure la primera vez que se pasa como argumento

Alcance Estático:
Variables y func/proc/sub que No son pasadas como args toman el valor de cuando se define/declara la rutina actual



(b) Alcance dinámico y asociación profunda

```

int x = 4, y = 1;
sub asta (int x) {
  y := 2 + x;
}
sub terraneo (int y, sub ida, sub urbio) {
  if (y < 8) {
    sub asta (int x) {
      x := y - 2;
    }
    terraneo (y + 8, asta, ida);
  } else if (y < 16) {
    terraneo (y + 8, urbio, asta);
  } else {
    int x = 4;
    ida (x + y);
    urbio (x + y);
  }
  print(x, y);
}

```

```

terraneo(x, asta, asta);
print(x, y);

```

Global

```

terraneo sub
  asta sub
    y 1
    x 4

```

Asociación Profunda y alcance Dinámico

Imprime:

```

4 26
4 12
4 4
4 1

```

X=3

Y=1

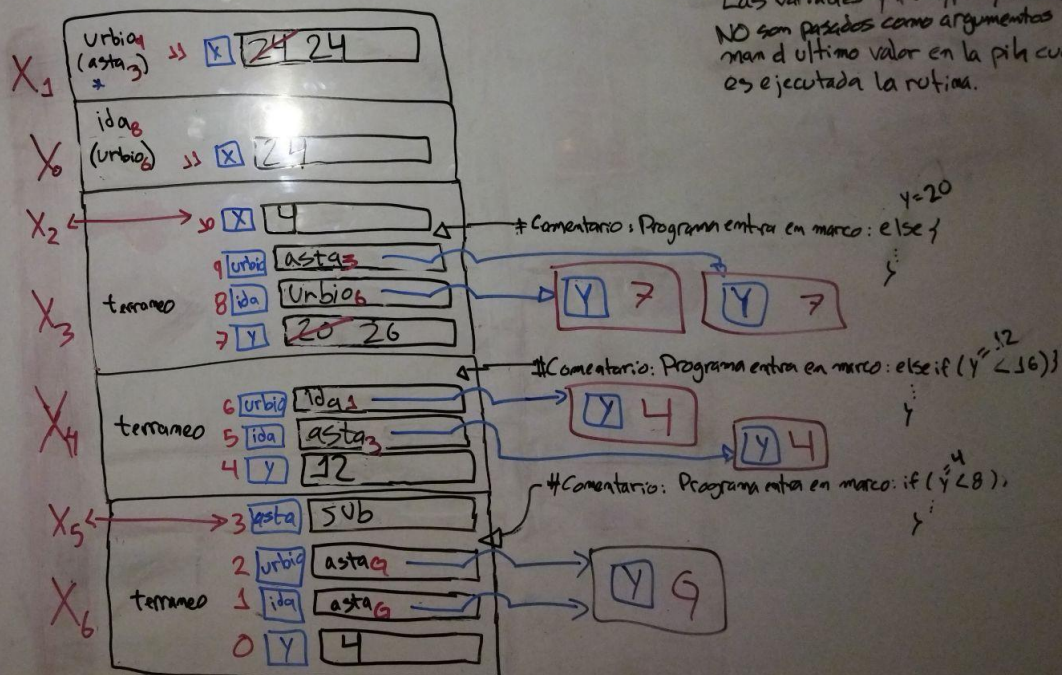
Z=3

Asociación Profunda:

Primera Clase - en el entorno de la def. creada. Se asocia la clausura en el cuerpo de la func.

Alcance Dinámico:

Las variables y func/proc/sub que NO son pasados como argumentos toman el último valor en la pila cuando es ejecutada la rutina.



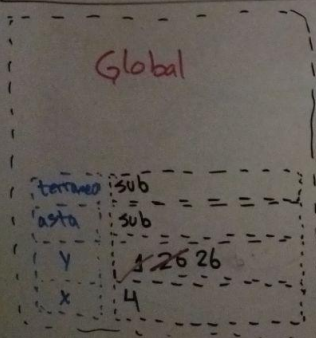
(c) Alcance estático y asociación superficial

```

int x=4 , y=1;
sub asta (int x) {
  y:=2+x;
}
sub terraneo (int y, sub ida, sub urbio) {
  if ( y < 8 ) {
    sub asta (int x) {
      x:=y-2;
      terraneo( y+8 , asta, ida);
    }
  } else if ( y < 16 ) {
    terraneo( y + 8 , urbio, asta);
  } else {
    int x= 4 ;
    ida(x+y);
    urbio(x+y);
  }
  print(x, y);
}

terraneo(x, asta, asta);
print(x, y);

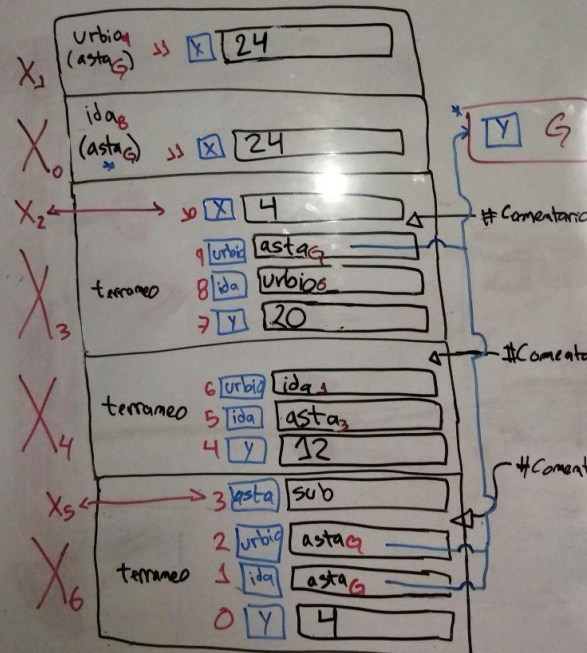
```



Asociación Superficial y alcance Estático

Imprime:

4	20
4	12
4	4
4	26



X=3

Y=1

Z=3

Asociación Superficial:

Ambiente de referencia cuando se invoca la func./momento de ejecución

Alcance Estático:

Variables y func/proc/sub que NO son pasadas como argumentos toman el valor de cuando se define/declara la rutina actual.

#Comentario: Programa entra en marco: else {
y=20
}

#Comentario: Programa entra en marco: else if (y=12)
}

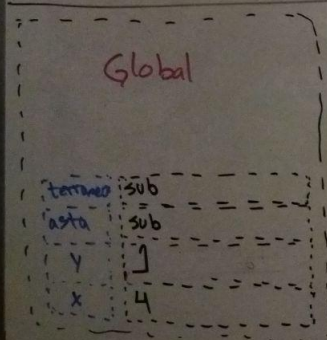
#Comentario: Programa entra en marco: if (y=4)
}

(d) Alcance dinámico y asociación superficial

```

int x=4 , y=1;
sub asta (int x){
  y:=2+x;
}
sub terraneo (int y, sub ida, sub urbio){
  if ( y < 8 ){
    sub asta (int x){
      x:=y-2
    }
    terraneo( y+8 , asta, ida);
  } else if ( y < 16 ){
    terraneo( y+8 , urbio, asta);
  } else {
    int x= 4 ;
    ida(x+y);
    urbio(x+y);
  }
  print(x, y)
}
terraneo(x, asta, asta);
print(x, y)

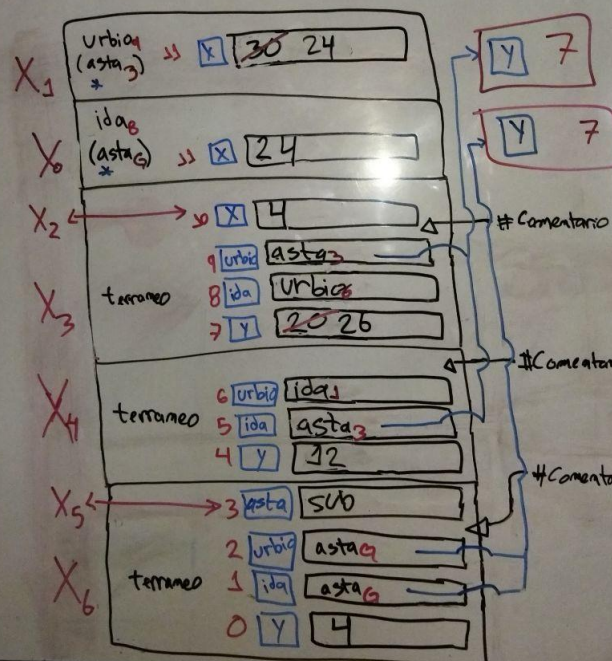
```



Asociación Superficial y alcance Dinámico

Imprime:

4	26
4	12
4	4
4	1



X=3
Y=1
Z=3

Asociación Superficial:

Ambiente de rep es cuando se invoca la función/momento de ejecución

Alcance Dinámico:

Las variables y func/proc/sub que NO son pasados como argumentos toman el último valor en la pila cuando es ejecutada la rutina.