Paradigmas de la Programación
Práctico 2: Fundamentos de lenguajes imperativos (y semántica)

Laura Alonso Alemany Ezequiel Orbe

27 de marzo de 2015

Seleccione la(s) respuesta(s) correcta(s)

1. La semántica de los lenguajes de programación debe reflejar formalmente...
   a) la intención del programador.
   b) los efectos de un programa en una computadora.
2. La semántica operacional...
   a) describe el significado de un programa como una secuencia de transiciones de estado a estado.
   b) usa una máquina abstracta para representar el significado de un programa.
   c) tiene un conjunto de operaciones para definir la semántica de un programa.
3. La semántica denotacional...
   a) explica el significado de un programa paso a paso.
   b) explica un programa como una función (parcial) de estados a estados.
   c) son una traducción de un programa en cualquier lenguaje a cualquier otro lenguaje.
4. Los activation records...
   a) muestran el estado de una máquina abstracta en un momento de la ejecución.
   b) se corresponden con los bloques en el texto del programa.
   c) se apilan en la pila de ejecución.
   d) todos los anteriores.
   e) ninguno de los anteriores.
5. El siguiente código es código espagueti porque...

```
10  IF  (X  .GT.  1)  GO TO  30
20  X = –X
      IF  (X  .LT.  1)  GO TO  60
30  IF  (X*Y  .LT.  1)  GO TO  40
      X = X–Y–Y
40   X = X+Y
60  CONTINUE
      X = A
      Y = B–A
      GO TO  20
```

a) tiene bloques.

b) las líneas están numeradas.

c) no podemos armar un árbol sintáctico del programa.

d) usa expresiones GOTO.

6. El identificador de una variable es...

   a) la dirección de memoria donde se encuentra un valor que se usa en un programa.

   b) el string que se usa en el texto de un programa para referirse a una variable.

   c) el tipo del valor de la variable.

7. Las variables están compuestas de... (elija la opción más precisa)

   a) un nombre y un valor.

   b) un l-valor y un r-valor.

   c) un identificador y un valor.

   d) un identificador que está ligado a un valor que está ligado a una dirección en memoria.

   e) un identificador que está ligado a una dirección en memoria que contiene un valor.

   f) una dirección de memoria, un valor y un identificador en el texto del programa.

8. Los "huecos" (regiones del programa en los que una variable no se puede acceder) en el alcance de una variable (identificador de variable) se dan porque...

   a) el identificador de la variable se usa para ligarse a otra variable.

   b) el lifetime de la variable no llega hasta ahí.

   c) tenemos un programa no estructurado.

9. En el siguiente programa, la segunda ocurrencia de P es libre o ligada?

```
declare P in
proc { P X }
  if X > 0 then { P X–1 } end
end
```

   a) libre

   b) ligada

Ejercicios prácticos

1. Dé informalmente (explicando con lenguaje cotidiano, sin necesidad de notación matemática) la semántica denotacional de estas dos funciones, es decir, explique informalmente qué es lo que hacen. Son equivalentes?

```
function foo(x) is:
  if predicate(x) then
    return foo(bar(x))
  else
    return baz(x)

function foo(x) is:
  while predicate(x) do:
    x <— bar(x)
  return baz(x)
```

2. Diagrame los estados de la pila de ejecución en los diferentes momentos de la ejecución del siguiente programa, mostrando cómo se apilan y desapilan los diferentes activation records a medida que se va ejecutando el programa. Puede representar los activation records con la información de variables locales y control links.

```
{int x=2 {int y=3; x=y+2;}}
```

3. Diagrame como en el ejercicio anterior, pero añada en el activation record la información necesaria para poder recuperar la información de la función, con *alcance estático*.

```
int sum(int n){
    if(n==0)
       return n;
    else
       return n+sum(n-1);
}
sum(4);
```

4. Diagrame como en el ejercicio anterior.

```
bool isPalindrome(char* s, int len)
{
  if(len < 2)
    return TRUE;
  else
    return s[0] == s[len-1] && isPalindrome(&s[1], len-2);
}
isPalindrome(``ada'')
```

5. Diagrame como en el ejercicio anterior.

```
fact(n) = if n<=1 then 1 else n*fact(n-1); fact(4);
```

6. Diagrame como en el ejercicio anterior, pero ahora con *alcance dinámico*.

```
var x=1;
function g(z) {return x+z;}
function f(y) {
  var x = y+1;
  return g(y*x);
}
f(3)
```

7. Explique el funcionamiento del siguiente programa con alcance estático y con alcance dinámico. Ayúdese de diagramas de active records si le resulta útil.

```
local D R Div in
    D=5
    proc {Div X ?Z}
       Z=X div D
    end
    {Div 10 R}
    {Browse R}
    local Y=0 R2 in
       {Div 10 R2}
       {Browse R2}
    end
 end
```

8. Diagrame como en el ejercicio anterior, pero en dos versiones: una con un compilador que NO optimice las llamadas a la cola y otra con un compilador que sí las optimice, sin una optimización específica para llamadas recursivas a la cola, es decir, sin *overlay*, pero sí consiguiendo que el tamaño de la pila se mantenga constante.

```
def fact(n,acc):
    if n==0:
      return acc
    return fact(n-1, acc*n)
fact(4,1)
```

9. El siguiente es un ejemplo de "*spaghetti code*". Reescríbalo de forma que NO use saltos (GOTO), y en cambio use programación estructurada.

```
10 i = 0
20 i = i + 1
30 PRINT i; " squared = "; i * i
40 IF i >= 10 THEN GOTO 60
50 GOTO 20
60 PRINT "Program Completed."
70 END
```