Contents

Parcial 1 2022-06-23.pdf

1)

1. **[10 pt.]** El siguiente código tiene una instrucción `break`. Esta instrucción produce un salto a otra parte del código. Entonces, el código que usa `break`, ¿es una característica del código spaghetti? ¿por qué?

```
1  while (true) {
2    decrease(k);
3    if (k!=0) {
4      set(p,u_k);
5      increasev2(a[k]);
6      if (q != 0) {
7        break;
8      }
9    } else {
10     return;
11   }
12 }
```

El uso de break no es una característica del codigo espaghetti, ya que al usar break no se produce un salto arbitrario en el el programa, si no que solo se sale del bloque del bucle que contenia al break.

Usando break es imposible saltar a un bloque que no se había ejecutado previamente, a diferencia de usar GOTO.

2)

2. Escriba en pseudocódigo un programa en el que:

   a) **[10 pt.]** una componente tenga efectos secundarios pero sea determinística

   b) **[10 pt.]** una componente no tenga efectos secundarios pero no sea determinística

a)

Asumiendo pasaje por referencia:

```
proc f(x) {
    x = 2
}
int x = 4
f(x)
print(x)
```

b)

```
int y = 5
fun f(x) {
    return x+y
}
int x = 3
int res1 = f(x)
y = 8
int res2 = f(x)
print(res1 != res2)
```

3) TODO

4)

4. **[10 pt.]** ¿Qué imprime el siguiente programa con alcance estático, y qué imprime con alcance dinámico? Justifique su respuesta usando los conceptos como el de *lugar de definición de la función* y el de *variable global*.

```
x : integer      -- global

procedure set_x(n : integer)
    x := n

procedure print_x
    write_integer(x)

procedure first
    set_x(1)
    print_x

procedure second
    x : integer
    set_x(2)
    print_x

set_x(0)
first()
print_x
second()
print_x
```

Estatico

print	que x se toma
1	global
1	global
2	global
2	global

Dinamico

print	que x se toma
1	global
1	global
2	definido en second
1	global

Explicacion

La principal diferencia entre el funcionamiento de los dos alcances se puede ver en la funcion `second`. En el alcance estatico, cuando se llama a la funcion `set_x`

dentro de second, se toma la definicion mas cercana a la declaracion de `set_x`, es decir, la variable global. En cambio, teniendo en cuenta el mismo contexto, en el alcance dinamico el valor de x se toma de la ultima vez que aparecio en el stack, es decir, de cuando se definio la variable local x adentro de `second`, por lo cual no se modifica la variable global y una vez se sale de la funcion `second` se tiene que el valor de la x global sigue siendo 1.

5)

5. **[20 pt.]** Cuáles son los valores de "x" y "z" al final del siguiente bloque, asumiendo que el pasaje de parámetros es a) por valor, b) por referencia y c) por valor-resultado?

```
{ int y;
  int z;
  y := 7;
  { int f(int x) {
      x := x+1;
      y := x;
      x := x+1;
      return y
    };
    int g(int x) {
      y := f(x)+1;
      x := f(y)+3;
      return x
    };
    z := g(y)
  }
}
```

a) Por valor

$y = \cancel{7 8 9} 10$

$z = g(y) = g(7) = 13$

$g(7)$ { $y = f(7) + 1 = 8 + 1 = 9$
$x = f(y) + 3 = f(9) + 3 = 10 + 3 = 13$ → ret

$f(7)$ { $x = 7 + 1 = 8$
$y = x = 8$ → return
$x = x + 1 = 9$

$f(9)$ { $x = 9 + 1 = 10$
$y = 10$ → return
$x = 10 + 1 = 11$

VF $y = 10$
$z = 13$

b) por referencia

$y = \cancel{7 8 9 10 11 12} 15$

$z = g(y) = g(7) = 15$

$g(7)$ { $y = f(7) + 1 = 9 + 1 = 10$ → $y = 10$
$x = f(y) + 3 = f(10) + 3 = 12 + 3 = 15$ → $y = 15$ ret

$f(7)$ { $x = 7 + 1 = 8$ → $y = 8$
$y = x = 8$ → $y = 8$
$x = x + 1 = 8 + 1 = 9$ → $y = 9$ → return

$f(10)$ { $x = 10 + 1 = 1$ → $y = 11$
$y = x = 11$ → $y = 11$
$x = x + 1 = 11 + 1 = 12$ → $y = 12$ → return

VF $y = 15$
$z = 15$

c) Por valor resultado

$y = \cancel{7 8 9 10 11} 13$

$z = g(y) = g(7) = 13$

(7) { $y = f(7) + 1 = 8 + 1 = 9$
$x = f(y) + 3 = f(9) + 3 = 10 + 3 = 13$ → ret
→ $y = 13$

(7) { $x = 7 + 1 = 8$
$y = x = 8$ → $y = 8$ → ret
$x = x + 1 = 8 + 1 = 9$
→ $y = 9$

$f(9)$ { $x = 9 + 1 = 10$
$y = x = 10$ → $y = 10$ → ret
$x = x + 1 = 10 + 1 = 11$
→ $y = 11$

VF $y = 13$
$z = 13$

6)

Explicacion

Se entra al primer bloque try, se entra al try anidado, se levanta una excepcion, esta excepcion es manejada por el catch del try anidado, se imprime `Inner_catch_block ...`, se ejecuta el bloque del finally, se imprime `Inner_finally_block`.

Como la excepcion lanzada no fue manejada no se ejecuta el catch del primer try. Como el finally siempre se ejecuta se imprime `Outer_finally_block`

Todo lo que se imprime

```
Inner_catch_block...
Inner_finally_block
Outer_finally_block
```

Parcial 1 2022-

1)

1. **[10 pt.]** Escriba la expresión que se representa en este árbol de tipos. ¿Hay algún error de tipos? Si lo hay, explique dónde se encuentra. Si no lo hay, escriba la signatura de tipos de la expresión.

$\lambda\ (g,h) = g(h) + 2$

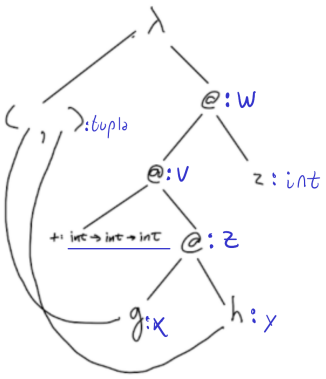
$g = t \to h$

$+ =$

① $x = y \to z$

② $int \to int \to int = z \to v$

③ $v = int \to w$



Por ② y ③

$int \to int \to int = z \to int \to w$

④ $z = int$

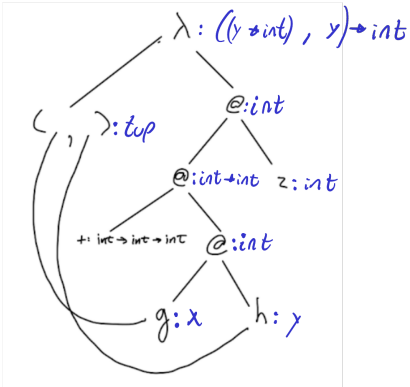
⑤ $w = int$

Por ① y ④

$x = y \to int$

Por ③ y ⑤

$v = int \to int$



jkjkj

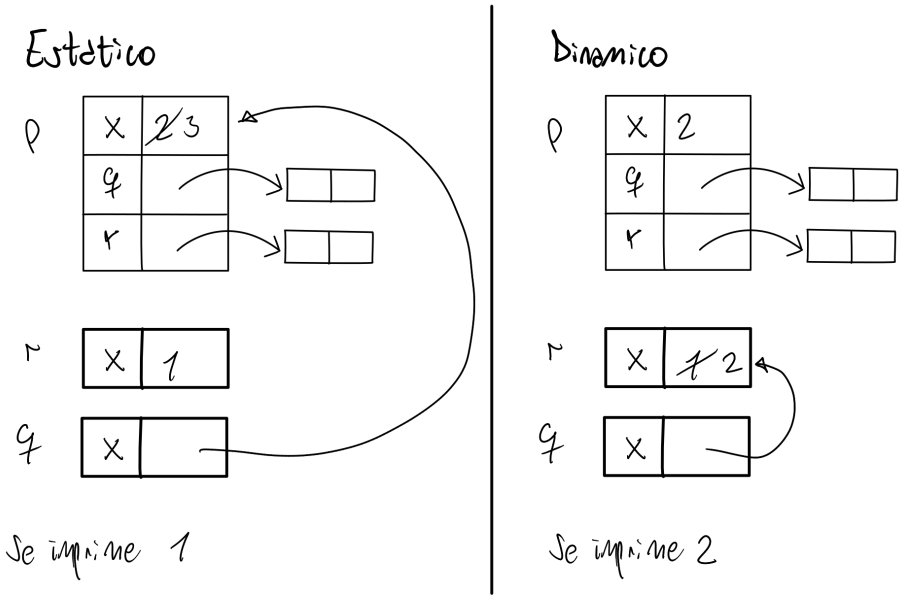
2)

No, el uso de break no es una característica del codigo spaghetti, ya que al ejecutarse el break se sale del bloque del bucle hacia un bloque previamente visitado, a diferencia de por ejemplo un GOTO, que salta arbitrariamente a una linea especifica, modificando el flujo normal del programa.

3)

3. **[10 pt.]** En el siguiente programa, ¿encontraremos una diferencia si el alcance del lenguaje es estático o si el alcance es dinámico? ¿Qué se imprimiría en cada caso?

```
1   procedure p;
2       x: integer;
3       procedure q;
4          begin x := x+1 end;
5       procedure r;
6          x: integer;
7          begin x := 1; q; write(x) end;
8       begin
9       x:= 2;
10      r
11      end;
```



La principal diferencia que podemos encontrar entre el alcance dinamico y el alcance estatico la podemos ver en el procedimiento `q`.

Utilizando alcance estatico, se busca el valor para una variable libre en la ubicacion mas cercana a la definicion de la funcion, por lo cual la variable `x` dentro de `q` hace referencia a la variable global `x`.

Por otro lado, para determinar el valor de una variable libre en el alcance dinamico se busca en la ultima aparicion de esa variable en el stack. Por lo cual, el `x` en el procedimiento `q` termina referenciando al `x` local definido dentro de `r`, ya que `r` fue quien llamó a `q`.

4)

4. **[10 pt.]** El siguiente texto explica lo que es un *thunk*.

> *A simple implementation of çall by name"might substitute the code of an argument expression for each appearance of the corresponding parameter in the subroutine, known as a "thunk".*

Escriba el *thunk* resultante de compilar en el siguiente programa, donde los argumentos se pasan mediante la estrategia *call-by-name*:

```
1  int i;
2  char array[3] = { 0, 1, 2 };
3
4  i = 0;
5  f(a[i]);
6
7  int f(int j)
8  {
9      int k = j;
10     i = 2;
11     k = j;
12 }
```

```
int f(int a[i])
{
    int k = a[i];
    i = 2;
    k = a[i];
}
```

5)

5. **[10 pt.]** El siguiente programa en C++, ¿es declarativo? ¿Por qué?

```
1  int values[4] = { 8, 23, 2, 4 };
2  int sum = SumArray(values);
3  RotateArrayIndices(values, -1);
```

No, no es declarativo, ya que segun como es utilizado `RotateArrayIndices` y por su nombre podemos intuir que recibe un array de valores y mueve el contenido de todo indice i a la posicion i-1, modificando así el array. Por lo cual, como modifica el array que se le pasa como argumento, quiere decir que produce efectos secundarios. Y como produce efectos secundarios podemos concluir que no es declarativo.

6)

```
1  class Program
2  {
3      static void Main(string[] args)
4      {
5          try
6          {
7              try
8              {
9                  throw new Exception("exception thrown from try block");
10             }
11             catch (Exception ex)
12             {
13                 Console.WriteLine("Inner catch block handling {0}.", ex.Message);
14                 throw;
15             }
16             finally
17             {
18                 Console.WriteLine("Inner finally block");
19                 throw new Exception("exception thrown from finally block");
20                 Console.WriteLine("This line is never reached");
21             }
22         }
23         catch (Exception ex)
24         {
25             Console.WriteLine("Outer catch block handling {0}.", ex.Message);
26         }
27         finally
28         {
29             Console.WriteLine("Outer finally block");
30         }
31     }
32 }
```

Se entra al bloque del primer try; se entra al bloque del try anidado; el try anidado lanza una excepcion; la excepcion es manejada por el catch del try anidado; Se ejecuta el bloque del finally del try anidado; se lanza una nueva excepcion; El catch del primer try maneja la excepcion lanzada en el finally del try anidado; Se ejecuta el bloque del finally del primer try;

7)

7. **[20 pt.]** Lea el siguiente texto y explique brevemente qué es una colisión de nombres. Atención: en clase hemos usado otro término para referirnos al mismo fenómeno. Si tenemos un lenguaje orientado a objetos, hay un contexto en el que podemos encontrar una mayor cantidad de colisiones de nombres, descríbalo.

   No todos los lenguajes orientados a objetos permiten el tipo de contextos con que es más fácil que suceda una colisión de nombres. Explique algún caso de un lenguaje con orientación a objetos que limitó su expresividad para evitar esos contextos.

Una colision de nombres sucede cuando hay mas de un metodo o funcion con el mismo nombre y el compilador no sabe a que elemento se está haciendo referencia, por lo cual produce un error.

El contexto en el que podemos encontrar una mayor colision de nombres es cuando se utiliza herencia multiple, ya que al heredar de varias clases es mas probable que se repita el nombre de alguna funcion.

Un lenguaje que limito su expresividad para evitar estos contextos es Java. El cual no permité la herencia multiple, si no que utiliza clases abstractas sin implementacion llamadas interfaces. Al no tener implementacion no seria gran problema que se utilicen dos interfaces con los mismos metodos, y por ende no habria problemas de colision.

Parcial 1 2019-06-18.pdf

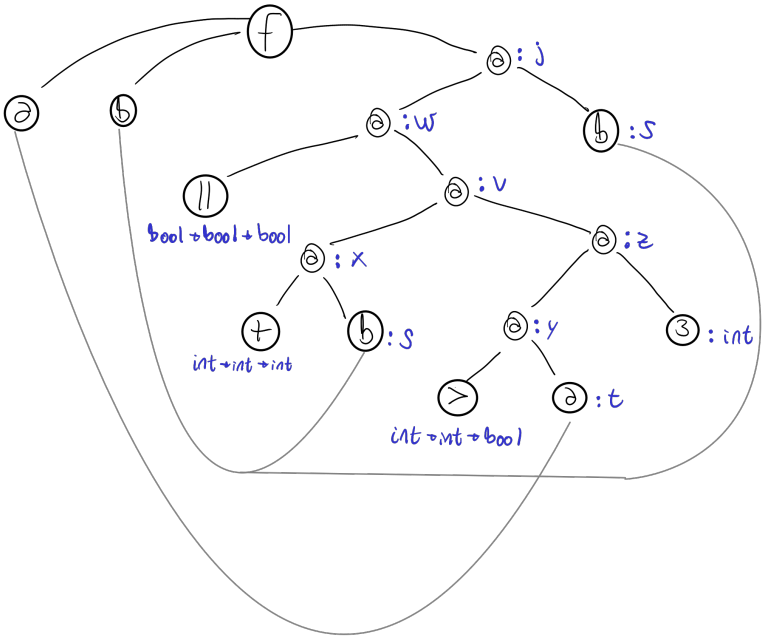
2)

2. **[10 pt.]** La siguiente expresión está mal tipada:

```
f (a, b) = ( b + a > 3 ) || b
```

Muestre con el árbol para inferencia de tipos dónde se encuentra el conflicto y en qué consiste. Cómo podría resolver este conflicto un lenguaje de tipado fuerte? y uno de tipado débil?


```
f (a, b) = ( b + a > 3 ) || b
```

① $int \to int \to int = S \to X$

② $int \to int \to bool = t \to y$

③ $Y = int \to z$

④ $X = z \to V$

⑤ $bool \to bool \to bool = V \to W$

⑥ $W = S \to j$

Por ② y ③:

$int \to int \to bool = t \to int \to z$

$\therefore$ ⑦ $t = int$, ⑧ $z = bool$

Por ①, ④, ⑧

$int \to int \to int = S \to bool \to V$

lo cual nos daría que $z = bool$ y $z = int$, como z tiene 2 tipos distintos quiere decir que hay un error de tipos

Un lenguaje de tipado fuerte probablemente produzca un error.

Un lenguaje de tipado debil seguro haria un casteo de bool a int si se suma con

un booleano o de int a bool, si se le hace un OR.