Diseño y Paradigmas de Lenguajes - Año 2015
Práctico Nro. 4
Control de secuencia y datos en subprogramas

Corresponde a Cap. VII Control de Subprogramas y Cap. XI Avances en el diseño de lenguajes pags: 375-404 del Pratt.

Nota: Los ejercicios marcados con ▶ son para pasarlos y ejecutarlos en máquina.

Ejercicio 1.

Considerando el siguiente código en lenguaje C, muestre gráficamente como van cambiando las estructuras de la activación de cada subprograma, en cada paso de la ejecución. Muestre los segmentos de códigos de los subprogramas y como se modifican la pila de ejecución y el heap. Tenga en cuenta que deben ser almacenados los valores correspondientes al *punto de regreso (retorno)*: (pi, pe) y los punteros CIP y CEP deben ser actualizados cuando corresponda.

```
int b = 5;

int M(int d){
 static int c = 1;
  c = c + d * b;
  return c;
}
void P(){
 int a = 0;
 if (b != 0){
     b = 0;
     P();
   }
  else a = 1;
}
void N(int* u){
 int* v;
  (*u) = b;
  v = (int*)malloc(sizeof(int));
  v = u;
  P();
}
void main(){
 int* t = (int*)malloc(sizeof(int));
 b = M(5)
 N(t);
}
```

Ejercicio 2.

Especifique los ambientes de referenciación locales y no locales de cada uno de los subprogramas, y mencione que variables son visibles en los bloques del while, del esqueleto de código C que se lista a continuación:

```
void eldoble(int); //prototipo
int a,e,f;
void main() {
 int x, y, z;
  while ( . . . ) {
    int a, b, x;
    . . .
    while ( . . . ) {
      int a, e;
      . . .
     }
   }
 x = 4;
 eldoble(x);
}
void eldoble(int b){
 int a,e;
 e = b * 2;
 a = a + 1;
 printf("El valor doble es %d:\n\n", e);
}
```

Ejercicio 3.

Especifique los ambientes de referenciación locales y no locales, de cada uno de los procedimientos y del programa principal del siguiente código en un lenguaje hipotético que soporta anidamientos.

```
procedure main(){
 var a,b,c;
 procedure p1(a){
     . . .
  }
 procedure p2(c){
  var b;
  procedure p3(){
     var b;
     . . .
     p4();
    }
   procedure p4(){
     var a;
     b = a + c;
    }
   . . .
   p1(b);
   p3();
  }
 . . .
 p2(b);
 }
```

Ejercicio 4.

Dado el código tipo Pascal que se lista a continuación:

```
Program Principal;
  type arreglo = array[0..5]of integer;
  var a:integer;
      b: real;
      c: arreglo = {2,4,6,8,10,12};
  Function P1(var a:integer, var b :integer): integer;
    var aa: real;
        c,d: integer;
    Procedure P2(d: integer);
      var bb, b: integer;
      begin
        b := 3;
        bb := d + a + c
      end;
    begin
      a := 2;
      b := 3;
      aa := 15.0;
      P1 := 1
    end;
  Function P4(var c:integer, var a:arreglo): integer;
    var aa:integer;
         d:integer;
    begin
      d := 5;
      aa := P1(d,d);
      a[1] := 2;
      c := c + 2;
      P4 := c
    end;
   begin
     a := 1;
     b := 1.0;
     a := P4(c[a],c)
   end.
```

Se pide:

1. Considerando los ambientes de referenciación de cada subprograma, responda a las siguientes preguntas justificando claramente sus respuestas:
   a) En el procedimiento **P2** ¿Es posible acceder a la variable **a** de Principal?
   b) En el procedimiento **P4** ¿Es posible acceder al parámetro formal **b** de **P1**?
   c) En el procedimiento **P4** ¿Es posible invocar al procedimiento **P2**?
2. ¿Existe algún punto del programa en el que dos o más variables se constituyan en *alias*? En caso afirmativo diga dónde y porqué.
3. Proponga al menos un alias, introduciendo modificaciones al código.

Ejercicio 6.

Considere el siguiente programa en lenguaje C:

```
int i = 0, a[5] = {1, 3, 1, 7, 9};
void swap(int a, int b) {
  int temp;
  temp = a;
  a = b;
  printf("%d, %d\n", a, a[i]);
  b = temp;
 }
void main() {
 swap(a[0],a[1]);
 i = 2;
 swap(i,a[i]);
}
```

Muestre el valor de cada una de las variables al finalizar cada una de las invocaciones de los subprogramas y los carteles que muestra. Considere que todos los parámetros son pasados:

1. Por referencia.
2. Por valor.
3. Por valor-resultado.
4. Por nombre.

Nota: Asuma que el lenguaje provee estos métodos de pasaje de parámetros.

Ejercicio 7.

Dado el siguiente fragmento de código en el lenguaje C:

```
int i = 2, j = 3, k[3][4];
k[0][0] = 9; k[0][1] = 8; k[0][2] = 7; k[0][3] = 6;  //fila 1
k[1][0] = 5; k[1][1] = 4; k[1][2] = 3; k[1][3] = 2;  //fila 2
k[2][0] = 1; k[2][1] = 2; k[2][2] = -1; k[2][3] = -2;  //fila 3

void p (int a, int b, int c) {
 a = 1; b = 0;
 while (b < 4){
   c = c * 5;
   b++;
  }
 }
int main() {
 p(i,j,k[i][j]);
}
```

a) Ejecute el programa, mostrando paso a paso la ejecución. Considere que C implementa los pasajes de parámetros por valor, referencia y nombre, y que *a* está pasado por valor, *b* por referencia y *c* por nombre.

b) Ejecute el programa, mostrando paso a paso la ejecución. Considere que todos los parámetros son pasados por referencia.

Ejercicio 8.

Para el código que se muestra a continuación en un lenguaje que soporta anidamientos y asumiendo que se utiliza una regla de alcance estático, se pide:

1. Muestre cómo queda la pila de ejecución en el caso en que la regla de alcance estático se implemente utilizando la cadena estática hasta la ejecución de subprograma *Q* inclusive.
2. Muestre cómo queda la pila de ejecución en el caso en que la regla de alcance estático se implemente utilizando el visualizador hasta la ejecución de subprograma *P* inclusive.
3. En el subprograma `Q` se accede a la variable `t`, calcule cómo se accedería a esta variable cuando se usa la cadena estática. Ahora suponga que se utiliza una regla de alcance estático implementada con el visualizador, ¿Cómo se accedería a la variable `t`? ¿Cuál de las dos implementaciones del alcance estático es más eficiente para el acceso de referencias no locales?

```
program Main;
 var a,b: integer;
     t,c: real;
     procedure P (a: integer, var w: integer);
       var t: real;
       procedure R (w: integer, var y: integer);
         var b: integer;
         procedure Q;
           begin
             t := c;
             w := 5;
           end                (* fin Q *)
         begin
           b := w;
           w := y;
           y := b;
           Q();
         end                  (* fin R *)
       begin
         w := a + 2;
         a := a - 2;
         R(w,a);
       end                    (* fin P *)
     procedure S(r: integer);
       var a: integer;
       begin
         a := r * 2;
         P(a,b);
       end                    (* fin S *)
 begin
   a := 3;
   b := 5;
   c := 2.5;
   S(b);
 end.
```

Ejercicio 9.

Dado el siguiente código de programa, tipo C:

```
void Main() {
 int a,b,c;
 ....
}

void Fun1(int c){
 int d;
 b = 10;
 ....
}

void Fun2(){
 int b,e;
 ....
}

void Fun3(){
 int c,a;
 ...
}
```

Dada cada una de las siguientes secuencias de llamadas, y asumiendo que se utiliza Alcance Dinámino:

a. Main invoca a Fun3; Fun3 invoca a Fun1.

b. Main invoca a Fun2; Fun2 invoca a Fun3; F3 invoca a Fun1.

Se pide:

1. ¿Qué variables son visibles durante la ejecución de la última función invocada?. Especifique (con cada variable) el nombre de la función en la que fue definida.
2. Implemente la regla de alcance dinámico utilizando la búsqueda directa en la pila de ejecución.
3. Muestre paso a paso la implementación de la regla de alcance dinámico que utiliza la Tabla Central y Pila Oculta de toda la ejecución del programa.
4. ¿Cómo se resuelve la referencia a la variable `b` en el procedimiento `Fun1`, cuando se utiliza la búsqueda directa? y ¿cuándo se utiliza la Tabla Central y Pila Oculta?

Ejercicio 10.

Considere el siguiente código en lenguaje Python:

```
a = 0
c = 8
def sub1():
        a = 5
        b = 7
        def sub2():
                global a
                c = 15
                print 'sub 2'
                print 'valor de a = ' + str(a)
                print 'valor de b = ' + str(b)
                print 'valor de c = ' + str(c)

        sub2()
        print 'sub 1'
        print 'valor de a = ' + str(a)
        print 'valor de c = ' + str(c)

def sub3(a,d):
      def sub4():
            e = 5
            print 'sub 4'
            print 'valor de e = ' + str(e)
            print 'valor de a = ' + str(a)
            print 'valor de c = ' + str(c)

      sub4()

sub1()
sub3(5,3)
```

Teniendo el cuenta que Python implementa la regla de alcance estático, ¿Cuál es la salida del programa? Compruebe su respuesta utilizando el intérprete Python.

Ejercicio 11.

Considere el siguiente código Java e indique los puntos en donde se producirá una *excepción*, especificando en qué casos se generará y donde se colocarán los manejadores para cada excepción.

```
import java.io.*;
public class Prueba {
   int Obtiene_entero(String str){
       return(Integer.parseInt(str));
   }

   void division(int i, int j){
     System.out.println("El resultado de la div:"+i/j);
   }

   void Procesa_arg(String[] args){
        int j;
        for(int i = 0;i <= args.length;i++){
           System.out.println(args[i]);
           j=Obtiene_entero(args[i]);
           division(j,i);
        }
    }

   void Muestra_ult_cadena(String str){
     System.out.println("El ultimo elemento es"+str.charAt(str.length()-1));
   }

   public static void main(String[] args) {
      String []a = new String[5];

      Procesa_arg(args);
      Muestra_ult_cadena(a[3]);
   }

}
```

Ejercicio 12.

Codifique un programa en el lenguaje Java, que permita el ingreso de números enteros finalizando con un número entero negativo y calcule la suma de los cuadrados de los valores ingresados. El programa debe emplear excepciones para asegurar que los valores sean enteros válidos y detectar el error cuando la suma de cuadrados supera el tamaño que una variable entera puede almacenar. En este caso de overflow (de la suma) se debe mostrar un mensaje de error y el programa deberá finalizar. Cuando se ingrese un número negativo que indica la finalización de la entrada de datos, se debe lanzar una excepción que muestre el resultado de la suma.

Ejercicio 13.

Explique y muestre gráficamente la ejecución y la implementación de las siguientes corutinas escritas en un lenguaje hipotético:

```
coroutine produce() {
    int q = 1
        while q is not 3
            resume consume()
            q++
}
coroutine consume(){
    int t = 4
        while t is not 1
            resume produce()
            t--
}

int main(){
    produce()
}
```

▶ **Ejercicio 14.**

Realice un programa en Java utilizando threads que implemente la multiplicación concurrente de una matriz por un vector. Este deberá generar la misma cantidad de threads que la cantidad de filas de la matriz y luego esperará que terminen todos los threads para mostrar el vector resultado. Cada thread realizará el producto de la fila que le corresponde con el vector y devolverá el resultado, también deberá mostrar por pantalla una identificación del thread.

Ejercicio Complementario

Dado el siguiente programa en un lenguaje hipotético, el cual permite anidamientos de subprogramas:

```
program Principal{
  var b,c,t: real;

  function P1: real{
    var j, t, z: real;

    if (b = 3.0) then
       b := 2.0; t := 1.0;
       z := P1()
    else
       c := 5.5; t := 0.0;
    return(t)
   }                                  //fin function P1

  procedure P3{
    var c, m: real;

    procedure P4 (m: real,c: real){
      static real t := 0.0;

      t := t + 1;
      m := m + b + t
     }                                 //fin procedure P4

    c := 8.0; m := 2.5;
    P4(m,c);
    c := P1();                        (*)
    m := m + 1;
   }                                  //fin procedure P3

  b := 3.0;
  P3();
}                                    //fin program Principal
```

Se pide:

a. Muestre gráficamente paso a paso como van cambiando las estructuras de la activación del programa hasta la sentencia marcada con (*) inclusive, considerando que la ejecución comienza en `program Principal`. Asuma para ello que el lenguaje utiliza una regla de alcance dinámico que se implementa a través de la Tabla Central y Pila Oculta.

b. En el programa existen varias declaraciones de la variable `c`, ¿a cuál de ellas corresponde la variable `c` que se utiliza en `P1` (considerando que se utiliza alcance dinámico)?

c. Ahora suponga que se utiliza una regla de alcance estático implementada con el visualizador, ¿Cómo se accedería a la variable `b` que se utiliza en `P4`?