LTP 2B - L1 - Examen primer parcial - 28 de Octubre de 2020

Alumno:

1 Cuestiones (40%)

1.1. (10%) Describe el mecanismo de paso de parámetros por valor (*call-by-value*). Ilústralo con un ejemplo.

Solución. Cuando se llama a una función f, por ejemplo con $f(e)$, donde e es una expresión, y se utiliza el paso de parámetros *por valor* (*call-by-value*), primero se obtiene el valor v de la expresión e y luego se invoca a la función f. El parámetro formal x correspondiente al argumento de la función f en su declaración queda vinculado al valor v y se utiliza en el cuerpo de la definición de f.

Ejemplo:

```
int f(int x) {
  return x+1;
}

void main() {
  int a = 1;

  println(f(a+1));
}
```

El número 3 se muestra por pantalla como resultado de la ejecución, ya que se le pasa a `f` el valor 2 (resultado de evaluar `a+1`) y luego se invoca a la función. El valor 2 queda vinculado al parámetro formal `x` de la función `f`. Durante la ejecución de `f` se incrementa su valor en una unidad y se devuelve como resultado: 3.

1.2. (10%) Describe las principales características del paradigma concurrente. Discute los problemas planteados por el uso de la concurrencia. Da al menos un ejemplo de lenguaje de programación que permita usar la concurrencia.

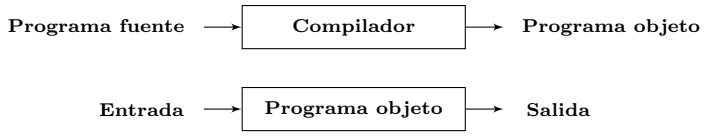
Solución. Los lenguajes de programación concurrentes se utilizan para programar la *ejecución* simultánea *de múltiples tareas*, que pueden consistir en un *conjunto de procesos* creados por un único programa. Problemas asociados a la concurrencia son: la *corrupción de los datos* compartidos, los *interbloqueos* entre procesos que comparten recursos, la *inanición* de un proceso que no consigue un recurso dado, el *indeterminismo* en el orden en el que se entrelazan las acciones de los distintos procesos. Ejemplos de lenguajes de programación que permiten la programación concurrente son *Java* y *Modula-2*.

1.3. (10%) ¿Cuál es el propósito del *análisis semántico* durante la compilación? Da ejemplos que ilustren su uso.

Solución. El análisis semántico es la etapa del proceso de compilación que sigue al análisis sintáctico. Parte del árbol sintáctico generado por éste y produce código intermedio, que es un código cercano al código máquina de cualquier procesador a partir del cual se podrá iniciar la generación de código de manera sistemática. En el análisis semántico se comprueban restricciones de la sintaxis que *no* pueden expresarse mediante la notación BNF pero que *sí* pueden comprobarse en tiempo de compilación. Por ejemplo, el uso de variables previo a su declaración, la compatibilidad y conversión de tipos en operaciones y asignaciones (*coercion*), la signatura de las funciones: los parámetros *reales* coinciden en número y tipo con los *formales*, etc.

1.4. (10%) Describe cómo se usan los *compiladores* en la implementación de lenguajes de programación. Da al menos dos ejemplos de lenguajes compilados.

Solución. El uso de programas en los lenguajes de programación 'compilados' tiene lugar en dos etapas. Primero se compila el programa y se obtiene un programa (o código) objeto completamente operativo. A continuación se utiliza dicho programa objeto para obtener los resultados (salida) a partir de los datos de entrada:



Ejemplos de lenguajes de programación cuyos programas se ejecutan siguiendo este esquema son C, C++, Fortran y Ada.

2 Problemas (60%)

2.1. (20%) Las siguientes clases de Java describen herramientas e instrumentos de ferretería que pueden almacenarse en diversos *recipientes* capaces de soportar un peso máximo medido en gramos. (...)

1. ¿Podría `protected` reemplazarse por `private` en la línea 22? ¿Y en la 20? Justifica cada respuesta.

   Solución. En la línea 22 de la definición de la clase `Recipient`, i.e.,

   ```
   22 protected long loadCapacity=0;
   ```

   no sería posible porque el atributo `loadCapacity` no podría utilizarse en la línea 40 del constructor de la subclase `Box`, i.e.,

   ```
   40 public Box() { loadCapacity = MAXBOX; }
   ```

   y tampoco en la línea 45 del constructor de la subclase `Tray`, i.e.,

   ```
   45 public Tray() { loadCapacity = MAXTRAY; }
   ```

   En cambio, el atributo `tools` declarado en la línea 20 de la definición de la clase `Recipient`, i.e.,

   ```
   20 protected Tool tools[ ];
   ```

   sí podra hacerse `private` porque no hay referencias a él en ninguna de las subclases de `Recipient`.

2. ¿Hay algún uso de la *sobrecarga*? ¿Y de la *coerción*? ¿Dónde?

   Solución. No hay sobrecarga, pero se utiliza *coerción implícita* en la línea 31, i.e.,

   ```
   31 loadCapacity = loadCapacity - tool.getWeight();
   ```

   para restar `tool.getWeight()` (de tipo `int`) de `loadCapacity` (de tipo `long`). En la línea 28, i.e.,

   ```
   28 if (tool.getWeight() > loadCapacity || i>MAXTOOLS) return false;
   ```

   se da un caso similar al comparar con el operador `>` los valores de `tool.getWeight()` y `loadCapacity`.

3. ¿Cuál de los siguientes grupos de instrucciones provocan un error? Si no es así, ¿cuál es el resultado (*true*/*false*) de `r.addPiece(t)` en cada caso? Justifica tus respuestas.

(1)
```
Recipient r = new Recipient();
Tool t = new Tool();
r.addPiece(t);
```

(2)
```
Recipient r = new Recipient();
Tool t = new Hammer();
r.addPiece(t);
```

(3)
```
Recipient r  = new Box();
Tool t = new Drill();
r.addPiece(t);
```

(4)
```
Recipient r = new Tray();
Hammer t = new Hammer();
r.addPiece(t);
```

(5)
```
Tray r = new Tray();
Tool t = new Tool();
r.addPiece(t);
```

(6)
```
Box r = new Recipient();
Tool t = new Hammer();
r.addPiece(t);
```

Solución.

- En (1) hay un error al intentar crear un objeto de la clase abstracta `Tool`.
- En (2) no hay error y el resultado de `r.addPiece(t)` es *false*, dado que `t.getWeight()` es 500 (pues `t` es de clase `Hammer`) y `loadCapacity` vale 0 para objetos (como `r`) de la clase `Recipient`.
- En (3) no hay error y el resultado de `r.addPiece(t)` es *true*, dado que `t.getWeight()` es 3000 (pues `t` es de clase `Drill`) y `loadCapacity` vale 100000 para objetos (como `r`) de la clase `Box`.
- En (4) no hay error y el resultado de `r.addPiece(t)` es *true*, dado que `t.getWeight()` es 500 (pues `t` es de clase `Hammer`) y `loadCapacity` vale 100000000 para objetos (como `r`) de la clase `Tray`.
- En (5) hay un error al intentar crear un objeto de la clase abstracta `Tool`.
- En (6) se produce un error al intentar asignar un objeto de la clase `Recipient` a la variable `r` de una subclase suya como `Box`.

4. Como parte de las llamadas a `r.addPiece(t)` en el código (1)-(6) anterior, ¿cuál es, en cada caso, el resultado devuelto por `tool.getWeight()` en la línea 28?

Solución.

- En (2) el resultado de `tool.getWeight())` es 500.
- En (3) el resultado de `tool.getWeight()` es 3000.
- En (4) el resultado de `tool.getWeight()` es 500.

2.2. (10%) Considera la siguiente clase `Main`: (...)

- ¿Hay algún tipo de uso de la *sobrecarga*? ¿Y de la *coerción*? ¿Dónde?

Solución.

– Respecto a la *sobrecarga*, en la línea 58, i.e.,

```
58 uWeight = uWeight + t.tools[i].getWeight();
```

se utiliza el operador de suma para números enteros largos y en la 59, i.e.,

```
59 System.out.println("Area:  "+uWeight);
```

se utiliza el operador de suma de cadenas de caracteres. También hay sobrecarga en el uso de `println` en las líneas 55 i.e.,

```
55 System.out.println(s.getNTools());
```

y 59 (ver arriba), ya que en el primer caso (línea 55) se aplica sobre argumentos enteros y en el segundo (línea 59) sobre cadenas de caracteres.

- Respecto a la *coerción*, en la línea 58 (ver arriba) se utiliza coerción implíta de enteros (`t.tools[i].getWeigth()`) a *enteros largos*. En la línea 59 (ver arriba) se utiliza coerción implícita de enteros largos (`uWeight`) a *strings*.

- ¿Podría la variable `t` de la línea 49 declararse de clase `Recipient` en vez de `Tray`? Justifica tu respuesta.

  Solución. La variable se declara en la línea 49 como sigue:

  ```
  49 private static Tray t = new Tray();
  ```

  y la variación propuesta sería

  ```
  49' private static Recipient t = new Tray();
  ```

  con lo que no habría diferencias en el comportamiento del programa, ya que no se utilizan atributos ni métodos específicos de `Tray` a través del objeto `t`.

- ¿Dónde se alojan los *objetos* referenciados por las variables `h`, `d`, y `t` (líneas 47-49)? ¿Y las *variables* `h`, `d` y `t`?

  Solución. Las declaraciones son las siguientes:

  ```
  47 private static Tool h = new Hammer();
  48 private static Tool d = new Drill();
  49 private static Tray t = new Tray();
  ```

  Los objetos se alojan en el *heap* mientras que las variables, al ser declaradas `static`, se alojan en el área de memoria estática.

2.3. (20%) El lenguaje de programación C incluye una *instrucción* `for` con el siguiente formato:

```
for (init ; test ; inc) inst
```

donde `init` inicializa una variable 'contador', `test` es una condición booleana que normalmente hace referencia a dicha variable, `inc` modifica el contador. Finalmente, `inst` es un bloque de instrucciones (*cuerpo del bucle*). Al principio de la ejecución se ejecuta la instrucción `init`. Las instrucciones del cuerpo del bucle se repiten mientras la condición `test` sea *cierta*. Tras cada ejecución de `inst` se ejecuta `inc`.

1. Realizar los cambios necesarios en la sintaxis de SIMP para poder utilizar dicha instrucción. ¿Qué nuevos símbolos *terminales* aparecen?

   Solución. Respecto a la sintaxis, solo cambia la definición BNF de instrucción como sigue:

   $$\begin{array}{rcl} i & ::= & \texttt{skip} \mid V \texttt{:=} a \mid i_1 \texttt{;}\, i_2 \mid \texttt{if}\ b\ \texttt{then}\ i_1\ \texttt{else}\ i_2 \mid \texttt{while}\ b\ \texttt{do}\ i \\ & & \mid \texttt{for (}i_1\ \texttt{;}\ b\ \texttt{;}\ i_2\texttt{)}\ i \end{array}$$

   Los nuevos símbolos terminales son: `for`, `(` y `)`.

2. Realizar los cambios necesarios en la semántica de paso pequeño y paso grande (*small-step*/*big-step*) de SIMP para poder ejecutar programas que incluyan la nueva instrucción.

   Solución.

- La semántica de paso pequeño de SIMP para la nueva instrucción podría definirse mediante la siguiente regla de *traducción*:

$$\langle \texttt{for (}i_1\ \texttt{;}\ b\ \texttt{;}\ i_2\texttt{)}\ i, s\rangle \to \langle i_1\texttt{; while}\ b\ \texttt{do (}i\ \texttt{;}\ i_2\texttt{)}, s\rangle$$

- La semántica de paso grande de SIMP para la nueva instrucción podría definirse mediante la siguiente regla:

$$\frac{\langle i_1, s\rangle \Downarrow s' \qquad \langle \texttt{while}\ b\ \texttt{do (}i\ \texttt{;}\ i_2\texttt{)}, s'\rangle \Downarrow s''}{\langle \texttt{for (}i_1\ \texttt{;}\ b\ \texttt{;}\ i_2\texttt{)}\ i, s\rangle \Downarrow s''}$$

3. Mostrar la traza de ejecución completa para el siguiente programa:

```
n := 3;
s := 0;
for (i := 1 ; i <= n ; i := i+2 )
   s := s + i
```

Solución.

(1) $\langle$`n := 3;s := 0; for (i := 1 ; i <= n ; i := i+2 ) s := s + i` $, \{\}\rangle \to$
$\quad\langle$`n := 3`$, \{\}\rangle \to$
$\quad\quad\langle 3, \{\}\rangle \Rightarrow 3$
$\quad\langle$`skip`$, \{n \mapsto 3\}\rangle$
(2) $\langle$`skip ; s := 0; for (i := 1 ; i <= n ; i := i+2 ) s := s + i` $, \{n \mapsto 3\}\rangle \to$
(3) $\langle$`s := 0;for (i := 1 ; i <= n ; i := i+2 ) s := s + i`$, \{n \mapsto 3\}\rangle \to$
$\quad\langle$`s := 0`$, \{n \mapsto 3\}\rangle \to$
$\quad\quad\langle 0, \{n \mapsto 3\}\rangle \Rightarrow 0$
$\quad\langle$`skip`$, \{n \mapsto 3, s \mapsto 0\}\rangle$
(4) $\langle$`skip ; for (i := 1 ; i <= n ; i := i+2 ) s := s + i`$, \{n \mapsto 3, s \mapsto 0\}\rangle \to$
(5) $\langle$`for (i := 1 ; i <= n ; i := i+2) s := s + i`$, \{n \mapsto 3, s \mapsto 0\}\rangle \to$
(6) $\langle$`i := 1 ; while i <= n do (s := s + i ; i := i+2)`$, \{n \mapsto 3, s \mapsto 0\}\rangle \to$
$\quad\langle$`i := 1`$, \{n \mapsto 3, s \mapsto 0\}\rangle \to$
$\quad\quad\langle 1, \{n \mapsto 3, s \mapsto 0\}\rangle \Rightarrow 1$
$\quad\langle$`skip`$, \{i \mapsto 1, n \mapsto 3, s \mapsto 0\}\rangle$
(7) $\langle$`skip ; while i <= n do (s := s + i ; i := i+2)`$, \{i \mapsto 1, n \mapsto 3, s \mapsto 0\}\rangle \to$
(8) $\langle$`while i <= n do (s := s + i ; i := i+2)`$, \{i \mapsto 1, n \mapsto 3, s \mapsto 0\}\rangle \to$
$\quad\langle$`i <= n`$, \{i \mapsto 1, n \mapsto 3, s \mapsto 0\}\rangle \Rightarrow$
$\quad\quad\langle$`i`$, \{i \mapsto 1, n \mapsto 3, s \mapsto 0\}\rangle \Rightarrow 1$
$\quad\quad\langle$`n`$, \{i \mapsto 1, n \mapsto 3, s \mapsto 0\}\rangle \Rightarrow 3$
$\quad$*true*
(9) $\langle$`s := s + i ; i := i+2 ; while i <= n do (s := s + i ; i := i+2)`$, \{i \mapsto 1, n \mapsto 3, s \mapsto 0\}\rangle \to$
$\vdots$
$\langle$`skip`$, \{i \mapsto 5, n \mapsto 3, s \mapsto 4\}\rangle \to$

2.4. (10%) ¿Es correcta la siguiente terna de Hoare? Justifica tu respuesta indicando los cálculos necesarios para obtenerla.

```
{ x < 0 }

  y := x+1;
  if x > y then z := x else z := y

{ 2z >= x+y }
```

Solución.

$$
\begin{aligned}
&wp(\texttt{y := x+1; if x > y then z := x else z := y},\ 2z \geq x + y) \\
&\quad = wp(\texttt{y := x+1}, wp(\texttt{if x > y then z := x else z := y},\ 2z \geq x + y)) \\
&\quad = wp(\texttt{y := x+1}, (x > y \land wp(\texttt{z := x}, 2z \geq x + y)) \lor (x \leq y \land wp(\texttt{z := y}, 2z \geq x + y))) \\
&\quad = wp(\texttt{y := x+1}, (x > y \land x \geq y) \lor (x \leq y \land y \geq x)) \\
&\quad = wp(\texttt{y := x+1}, x > y \lor x \leq y) \\
&\quad = wp(\texttt{y := x+1}, \mathit{true}) \\
&\quad \Leftrightarrow \mathit{true}
\end{aligned}
$$

Dado que $x < 0 \Rightarrow \mathit{true}$ siempre se cumple, la terna es correcta.