Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3641 – Lenguajes de Programación I
Enero–Marzo 2021

Examen 1

(30 puntos)

A continuación encontrará 6 preguntas (y una sorpresa al final), cada una de las cuales tiene un valor de 5 puntos. Sea lo más detallado y preciso posible en sus razonamientos y procedimientos.

En algunas preguntas, se usarán las constantes X, Y y Z. Estas constantes debe obtenerlas de los últimos tres números de su carné. Por ejemplo, si su carné es 09-40325, entonces $X = 3$, $Y = 2$ y $Z = 5$.

En aquellas preguntas donde se le pida decir qué imprime un programa, incluya los pasos relevantes de la ejecución del mismo con los cuales usted pudo alcanzar su conclusión.

En aquellas preguntas donde se le pida implementar un programa, mantenga su código en un repositorio `git` remoto (preferiblemente `Github`) y coloque un enlace al mismo en lugar de su respuesta. Todo su código debe ser legible y estar debidamente documentado.

La entrega se realizará por correo electrónico a rmonascal@gmail.com hasta las 11:59pm. VET del Viernes 19 de Agosto de 2021.

1. Escoja algún lenguaje de programación de alto nivel y de propósito general cuyo nombre empiece con la misma letra que su nombre (por ejemplo, si su nombre es "Pedro", podría escoger "Perl", "PLI", "Python", etc.).
   (a) De una breve descripción del lenguaje escogido.
      i. Diga qué tipo de alcances y asociaciones posee, argumentando las ventajas y desventajas de la decisión tomada por los diseñadores del lenguaje, en el contexto de sus usuarios objetivos.
      ii. Diga que tipo de módulos ofrece (de tenerlos) y las diferentes formas de importar y exportar nombres.
      iii. Enumere y explique las estructuras de control de flujo que ofrece.
      iv. Diga en qué orden evalúan expresiones y funciones
   (b) Implemente los siguientes programas en el lenguaje escogido:
      i. Dado un entero no–negativo n, calcular el factorial de n.

$$fact(n) = \left\{ \begin{array}{ll} 1 & \text{si } n = 0 \\ n \times fact(n-1) & \text{si } n > 0 \end{array} \right\}$$

      ii. Dadas dos matrices A y B (cuyas dimensiones son $N \times M$ y $M \times P$, respectivamente), calcular su producto $A \times B$ (cuya dimensión es $N \times P$).

$$\forall i \in [1..N], j \in [1..P] : (A \times B)_{i,j} = \sum_{k \in [1..M]} A_{i,k} \times B_{k,j}$$

2. Considere el siguiente programa escrito en pseudo-código:

```
int x = X + 1, y = Y;

proc ohno(int x) {
   y := 2 * x;
}

proc ohwell(int y, proc waitwhat) {
   if (y < 2 * (X + 1)) {
      proc ohno(int x) {
         x := y * 2;
      }
      ohwell(y + 2 * (X + 1), waitwhat);
   } else if (y < 4 * (X + 1)) {
      ohwell(y + 2 * (X + 1), ohno);
   } else {
      int x = Z;
      waitwhat(x + y);
   }
   print(x, y)
}

ohwell(x, ohno);
print(x, y)
```

Note que deberá reemplazar los valores para X, Y y Z como fue explicado en los párrafos de introducción del examen.

Diga qué imprime el programa en cuestión, si el lenguaje tiene:

(a) Alcance estático y asociación profunda

(b) Alcance dinámico y asociación profunda

(c) Alcance estático y asociación superficial

(d) Alcance dinámico y asociación superficial

Recuerde mostrar los pasos de su ejecución (por lo menos al nivel de cada nuevo marco de pila creado).

3. Se desea que modele e implemente, en el lenguaje de su elección, un programa que simule un manejador de memoria que implementa el *buddy system*. Este programa debe cumplir con las siguientes características:

   (a) Al ser invocado, recibirá como argumento la cantidad de bloques de memoria que manejará.

   (b) Una vez iniciado el programa, pedirá repetidamente al usuario una acción para proceder. Tal acción puede ser:

      i. `RESERVAR <nombre> <cantidad>`
      Representa una reserva de espacio de `<cantidad>` bloques, asociados al identificador `<nombre>`.
      El programa debe reportar un error e ignorar la acción si `<nombre>` ya tiene memoria reservada o no hay un espacio libre contiguo suficientemente grande como para satisfacer la petición (en cualquier caso, el mensaje de error debe ser claro e informativo).

      ii. `LIBERAR <nombre>`
      Representa una liberación del espacio que contiene el identificador `<nombre>`.
      El programa debe reportar un error e ignorar la acción si `<nombre>` no tiene memoria reservada (el mensaje de error debe ser claro e informativo).

      iii. `MOSTRAR`
      Debe mostrar una representación gráfica (en texto) de las listas de bloques libres, así como la información de nombres y la memoria que tienen asociada a los mismos.

      iv. `SALIR`
      Debe salir del simulador.

   Al finalizar la ejecución de cada acción, el programa deberá pedir la siguiente acción al usuario.

   Investigue herramientas para pruebas unitarias y cobertura en su lenguaje escogido y agregue pruebas a su programa que permitan corroborar su correcto funcionamiento. Como regla general, su programa debería tener una cobertura (de líneas de código y de bifuración) mayor al 80%.

4. Considere los siguientes iteradores, escritos en Python:

   (a) El iterador `zip`:

```
def zip(a, b):
   if a and b:
      yield (a[0], b[0])
      for p in zip(a[1:], b[1:]):
         yield p
```

   Considere también el siguiente fragmento de código que hace uso del iterador 'zip':

```
for p in zip([1, 2, 3], ['a', 'b', 'c']):
   print p
```

   i. Ejecute, paso a paso, el fragmento de código mostrado (por lo menos al nivel de cada nuevo marco de pila creado) y muestre lo que imprime.

   ii. Modifique el iterador `zip` de tal forma que obtenga un nuevo iterador `zipWith`, que recibirá como tercer argumento la función que usará para unir cada par de elementos.

   Por ejemplo, considere el siguiente fragmento de código:

```
for p in zipWith([0, 1, 2, 1], [1, 2, 1, 0], lambda x, y: x + y):
   print p
```

   Éste deberá generar:

   - $0 + 1 = 1$
   - $1 + 2 = 3$
   - $2 + 1 = 3$
   - $1 + 0 = 1$

   Note que ahora `zip` es un caso especial de `zipWith`, donde la operación que se realiza entre los elementos de la lista es `lambda x, y: (x, y)`.

(b) El iterador `misterio`, que hace uso de `zipWith`:

```
def misterio(p):
   yield p
   acum = []
   for p in zipWith([0, *p], [*p, 0], lambda x, y: x + y):
      acum += [p]
   for m in misterio(acum):
      yield m
```

Considere también el siguiente fragmento de código que hace uso del iterador `misterio`:

```
for m in misterio([1]):
   print p
```

i. Ejecute, paso a paso, el fragmento de código mostrado (por lo menos al nivel de cada nuevo marco de pila creado) y muestre lo que imprime.
Deberá reportar **únicamente** los primeros 7 elementos que imprime y dar un estimado sobre la cantidad de elementos que imprimiría el fragmento de código si se le permitiese continuar con su ejecución.
Pista: Los elementos generados por este iterador serán listas.

ii. Explique, a grandes rasgos, cómo funciona el iterador `misterio` y qué colección de elementos conocida está generando. Diga cómo aprovecha el iterador `zipWith` para generar la colección deseada.

iii. Modifique el iterador `misterio` de tal forma que obtenga un nuevo iterador `supenso`, que devolverá cada elemento de las listas generadas por `misterio` de forma *aplanada*.
Por ejemplo, si al ejecutar `misterio` resultan generadas las listas:
   - `[a, b, c]`
   - `[d]`
   - `[e, f]`

   El nuevo iterador `suspenso` deberá generar:
   - `a`
   - `b`
   - `c`
   - `d`
   - `e`
   - `f`

5. Considere la siguiente definición para una familia de funciones:

$$F_{\alpha,\beta}(n) = \left\{ \begin{array}{ll} n & \text{si } 0 \leq n < \alpha \times \beta \\ \displaystyle\sum_{i=1}^{\alpha} F_{\alpha,\beta}(n - \beta \times i) & \text{si } n \geq \alpha \times \beta \end{array} \right\}$$

Notemos que $F_{2,1}$ corresponde a la definición para los números de Fibonacci:

$$F_{2,1}(n) = \left\{ \begin{array}{ll} n & \text{si } 0 \leq n < 2 \\ F_{2,1}(n-1) + F_{2,1}(n-2) & \text{si } n \geq 2 \end{array} \right\}$$

Como un segundo ejemplo, $F_{3,4}$ corresponde a:

$$F_{3,4}(n) = \left\{ \begin{array}{ll} n & \text{si } 0 \leq n < 12 \\ F_{3,4}(n-4) + F_{3,4}(n-8) + F_{3,4}(n-12) & \text{si } n \geq 12 \end{array} \right\}$$

Tomando como referencia las constantes X, Y y Z planteadas en los párrafos de introducción del examen, definamos:

- $\alpha = ((X + Y) \bmod 5) + 3$
- $\beta = ((Y + Z) \bmod 5) + 3$

Se desea que realice implementaciones, en el lenguaje C de:

(a) Una subrutina recursiva que calcule $F_{\alpha,\beta}$ para los valores de α y β obtenidos con las fórmulas mencionadas anteriormente. Esta implementación debe ser una traducción directa de la fórmula resultante a código.

(b) Una subrutina recursiva **de cola** que calcule $F_{\alpha,\beta}$.

(c) La conversión de la subrutina anterior a una versión iterativa, mostrando claramente cuáles componentes de la implementación recursiva corresponden a cuáles otras de la implementación iterativa.

Realice también un análisis comparativo entre las tres implementaciones realizadas, mostrando tiempos de ejecución para diversos valores de entrada y ofreciendo conclusiones sobre la eficiencia. Es recomendable que se apoye en herramientas de visualización de datos (como los *plots* de Matlab, R, Octave, Excel, etc.)

6. Se desea que modele e implemente, en el lenguaje de su elección, un programa que maneje expresiones aritmáticas sobre enteros. Este programa debe cumplir con las siguientes características:

   (a) Debe saber tratar expresiones escritas en orden *pre–fijo* y *post–fijo*, con los siguientes operadores:

   - **suma**: Representada por el símbolo `+`.
   - **resta**: Representada por el símbolo `-`.
   - **multiplicación**: Representada por el símbolo `*`.
   - **división entera**: Representada por el símbolo `/`.

   (b) Una vez iniciado el programa, pedirá repetidamente al usuario una acción para proceder. Tal acción puede ser:

   i. `EVAL <orden> <expr>`
   Representa una *evaluación* de la expresión en `<expr>`, que está escrita de acuerdo a `<orden>`.
   El `<orden>` solamente puede ser:
   - `PRE`: Que representa expresiones escritas en orden *pre–fijo*.
   - `POST`: Que representa expresiones escritas en orden *post–fijo*.

   Por ejemplo:
   - `EVAL PRE + * + 3 4 5 7` deberá imprimir `42`.
   - `EVAL POST 8 3 - 8 4 4 + * +` deberá imprimir `69`.

   ii. `MOSTRAR <orden> <expr>`
   Representa una *impresión en orden in–fijo* de la expresión en `<expr>`, que está escrita de acuerdo a `<orden>`.
   El `<orden>` sigue el mismo patrón que en el punto anterior.
   Su programa debe tomar la precedencia y asociatividad estándar, donde:
   - La suma y la resta tienen la misma precedencia.
   - La multiplicación y la división entera tienen la misma precedencia.
   - La multiplicación y la división entera tienen mayor precedencia que la suma y la resta.
   - Todos los operadores asocian a izquierda.

   La expresión resultante debe tener la menor cantidad posible de paréntesis, de tal forma que la expresión mostrada como resultado tenga la misma semántica que la expresión que fue pasada como argumento a la acción.
   Por ejemplo:
   - `MOSTRAR PRE + * + 3 4 5 7` deberá imprimir `(3 + 4) * 5 + 7`.
   - `MOSTRAR POST 8 3 - 8 4 4 + * +` deberá imprimir `(8 - 3) + 8 * (4 + 4)`.

   iii. `SALIR`
   Debe salir del programa.

   Al finalizar la ejecución de cada acción, el programa deberá pedir la siguiente acción al usuario.

   Investigue herramientas para pruebas unitarias y cobertura en su lenguaje escogido y agregue pruebas a su programa que permitan corroborar su correcto funcionamiento. Como regla general, su programa debería tener una cobertura (de líneas de código y de bifuración) mayor al 80%.

7. RETO EXTRA: *GOLF!*

Considere el coeficiente binomial:

$$\binom{n}{m} = \frac{n!}{m! \times (n-m)!}$$

Considere también la función para hallar el n–ésimo número en la secuencia de fibonacci:

$$fib(n) = \left\{ \begin{array}{ll} n & \text{si } 0 \leq n < 2 \\ fib(n-1) + fib(n-2) & \text{si } n \geq 2 \end{array} \right\}$$

Finalmente, considere el logaritmo en base 2:

$$log_2(n) = m \Leftrightarrow 2^n = m$$

Definiremos la función *jaweno* como:

$$jaweno(n) = \left\lfloor \log_2 \binom{fib(n+1)}{fib(n)} \right\rfloor$$

Sabiendo que $\lfloor x \rfloor$ es el piso de x (el mayor valor entero que es menor o igual a x)

Desarrolle, en el lenguaje de su elección, un programa que:

- Reciba por argumentos del sistema un valor para n, tal que $n \geq 0$ (esto puede suponerlo, no tiene que comprobarlo).
- Imprima el valor de $jaweno(n)$.

Su programa debe imprimir el valor correcto y tomando menos de 1 segundo de ejecución, por lo menos hasta $n = 20$.

Reglas del reto: Intente desarrollar su programa con la menor cantidad de caracteres posibles (incluyendo espacios en blanco).

- El ganador del reto tendrá 5 puntos extras.
- El segundo lugar tendrá 3 puntos extras.
- El tercer lugar tendrá 1 punto extra.

R. Monascal / Febrero 2021