

Examen 2

(30 puntos)

A continuación encontrará 5 preguntas (y una sorpresa al final), cada una de las cuales tiene un valor de 6 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 Domingo 19 de Noviembre de 2023.

1. Escoja algún lenguaje de programación de alto nivel y de propósito general cuyo nombre empiece con la misma letra que su apellido (por ejemplo, si su apellido es “Rodríguez”, podría escoger “Ruby”, “Rust”, “R”, etc.).

(a) De una breve descripción del lenguaje escogido.

- i. Enumere y explique las estructuras de control de flujo que ofrece.
- ii. Diga en qué orden evalúan expresiones y funciones
- iii. Diga qué tipos de datos posee y qué mecanismos ofrece para la creación de nuevos tipos (incluyendo tipos polimórficos de haberlos).
- iv. Describa el funcionamiento del sistema de tipos del lenguaje, incluyendo el tipo de equivalencia para sus tipos, reglas de compatibilidad y capacidades de inferencia de tipos.

(b) Implemente los siguientes programas en el lenguaje escogido:

- i. Defina un tipo de datos recursivo que represente numerales de Church.

A continuación un ejemplo en Haskell:

```
data Church = Cero | Suc Church
```

Recuerde que un numeral de Church se construye a partir de:

- Un valor constante que representa al cero.
- Una función *sucesor* que, para cualquier número n , devuelve $n + 1$.

Sobre este tipo se desea que implemente las funciones **suma** y **multiplicación**.

*Nota: No busque implementaciones online, ya que la mayoría no hace lo que se pide. No de un valor funcional para **Zero** ni para **Suc**. Ambos deben ser constructores de tipos y nada más.*

- ii. Defina un árbol binario con información en ramas y hojas.

A continuación un ejemplo en Haskell:

```
data Arbol a = Hoja a | Rama a (Arbol a) (Arbol a)
```

Sobre este árbol, defina una función **esMaxHeapSimétrico** que diga si el árbol en cuestión es un *max-heap* simétrico o no.

Recuerde que un *max-heap* es un árbol binario tal que, para cada rama, el valor almacenado en la misma es *mayor o igual* que todos los valores que se encuentran en ambos sub-árbol *hijos*.

Consideraremos que un *max-heap* es *simétrico* sí y sólo si ls recorridos en *pre-order* y en *post-order* del árbol producen la misma secuencia.

2. Se desea que modele e implemente, en el lenguaje de su elección, un programa que maneje expresiones aritméticas 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 bifurcación) mayor al 80%.

3. Considere los siguientes iteradores, escritos en Python:

(a) El iterador `misterio`:

```
def misterio(a, b, c, d):
    if c == 0:
        yield a
        for x in misterio(b, a, b, d - 1):
            yield x
    elif d > 0:
        for x in misterio(a, b + 1, c - 1, d):
            yield x
```

Tomando como referencia las constantes X , Y y Z planteadas en los párrafos de introducción del examen, considere también el siguiente fragmento de código que hace uso del iterador `misterio`:

```
a = 2 * X + 3 * Y + 2
b = 4 * Y + 5 * Z + 1
c = 5 * X + 2 * Z + 3
d = (a + b + c) % 7
for x in misterio(0, 1, 0, d + 1):
    print x
```

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.

(b) Dada una lista de enteros *con elementos únicos* (no se repiten), queremos un iterador que devuelva todas las sublistas *crecientes* que contiene:

Por ejemplo, para la lista `[1,4,3,2,5]`, las sublistas crecientes que continene son:

```
[]
[1]
[1,4]
[1,4,5]
[1,3]
[1,3,5]
[1,2]
[1,2,5]
[1,5]
[4]
[4,5]
[3]
[3,5]
[2]
[2,5]
[5]
```

Implemente este iterador en el lenguaje de su preferencia.

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

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

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

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

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

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

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 *imperativo* de su elección:

- (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.

Debe usar el mismo el lenguaje para estos tres ejercicios y asegurarse que su lenguaje tenga las estructuras de control de flujo necesarias para realizarlos (su lenguaje escogido debe, por tanto, ser imperativo).

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.)

5. Se desea que modele e implemente, en el lenguaje de su elección, un programa que simule un manejador de tipos de datos. Este programa debe cumplir con las siguientes características:
- (a) Debe saber manejar tipos atómicos, registros (**struct**) y registros variantes (**union**).
 - (b) Una vez iniciado el programa, pedirá repetidamente al usuario una acción para proceder. Tal acción puede ser:
 - i. **ATOMICO** <nombre> <representación> <alineación>
Define un nuevo tipo atómico de nombre <nombre>, cuya representación ocupa <representación> bytes y debe estar alineado a <alineación> bytes.
Por ejemplo: **ATOMICO char 1 2** y **ATOMICO int 4 4**
El programa debe reportar un error e ignorar la acción si <nombre> ya corresponde a algún tipo creado en el programa.
 - ii. **STRUCT** <nombre> [<tipo>]
Define un nuevo registro de nombre <nombre>. La definición de los campos del registro viene dada por la lista en [<tipo>]. Nótese que los campos no tendrán nombres, sino que serán representados únicamente por el tipo que tienen.
Por ejemplo: **STRUCT foo char int**
El programa debe reportar un error e ignorar la acción si <nombre> ya corresponde a algún tipo creado en el programa o si alguno de los tipos en [<tipo>] no han sido definidos.
 - iii. **UNION** <nombre> [<tipo>]
Define un nuevo registro variante de nombre <nombre>. La definición de los campos del registro variante viene dada por la lista en [<tipo>]. Nótese que los campos no tendrán nombres, sino que serán representados únicamente por el tipo que tienen.
Por ejemplo: **UNION bar int foo int**
El programa debe reportar un error e ignorar la acción si <nombre> ya corresponde a algún tipo creado en el programa o si alguno de los tipos en [<tipo>] no han sido definidos.
 - iv. **DESCRIBIR** <nombre>
Debe dar la información correspondiente al tipo con nombre <nombre>. Esta información debe incluir, tamaño, alineación y cantidad de bytes desperdiciados para el tipo, si:
 - El lenguaje guarda registros y registros variantes *sin empaquetar*.
 - El lenguaje guarda registros y registros variantes *empaquetados*.
 - El lenguaje guarda registros y registros variantes *reordenando los campos de manera óptima* (minimizando la memoria, sin violar reglas de alineación).El programa debe reportar un error e ignorar la acción si <nombre> no corresponde a algún tipo creado en el programa.
 - v. **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 bifurcación) mayor al 80%.

6. RETO EXTRA: ¡POLÍGLOTA!

Considere la misma función *wadefoc*, definida en el parcial anterior:

$$wadefoc(n) = fib(\lfloor \log_2(N_{n+1,n-1}) \rfloor + 1)$$

Decimos que un programa es *políglota* si el mismo código fuente puede ser compilado/interpretado por al menos dos diferentes lenguajes de programación.

Desarrolle un programa políglota que:

- Reciba por *la entrada estándar* un valor para n , tal que $n \geq 0$ (esto puede suponerlo, no tiene que comprobarlo).
- Imprima el valor de *wadefoc*(n).

Su programa debe imprimir el valor correcto y tomando menos de 1 segundo de ejecución, por lo menos hasta $n = 100$ en todos los lenguajes de programación considerados.

Reglas del reto: Intente desarrollar su programa de tal forma que la mayor cantidad de lenguajes de programación puedan compilarlo/interpretarlo. Debe indicar todos los lenguajes para los cuales su código fuente funciona y proporcionar instrucciones para ejecutarlo en cada uno de estos (que puede ser sencillamente un enlace a alguna herramienta online para interpretar el lenguaje, como tio.run o ideone.com)

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