Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3641 – Lenguajes de Programación I
Septiembre–Diciembre 2024

Examen 3

(20 puntos)

A continuación encontrará 4 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 Miércoles 20 de Noviembre de 2024.

1. Escoja algún lenguaje de programación de alto nivel y de propósito general que tenga la misma cantidad de caracteres que su primer nombre (por ejemplo, si su nombre es "Carlos", podría escoger "Python", "Prolog", "Modula", etc.).

   (a) De una breve descripción del lenguaje escogido.

      i. Diga qué tipos de datos posee y qué mecanismos ofrece para la creación de nuevos tipos (incluyendo tipos polimórficos de haberlos).

      ii. 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.

      iii. Diga si la totalidad de los tipos de datos del lenguaje conforman un álgebra. Diga si posse construcciones que corresponden a: el tipo producto, el tipo suma, el tipo cero (neutro de la suma) y el tipo uno (neutro del producto).

   (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 tipo que represente conjuntos de personas, donde cada persona tiene un `nombre` y una `edad`.

      Sobre este tipo, defina funciones que devuelvan:

      - La cantidad de personas en el conjunto.
      - El subconjunto de personas que son mayores de edad (cuya `edad` es mayor o igual a 18).
      - El nombre más común en el conjunto (si hay varios nombres con la máxima cantidad de ocurrencias, se puede tomar cualquiera de ellos).

2. Tomando en cuenta las definiciones de X, Y y Z planteadas en los párrafos de introducción del examen, considere las siguientes definiciones de variables:

   - $L_1 = min(X, Y)$
   - $L_2 = min(X, Z)$
   - $L_3 = min(Y, Z)$
   - $U_1 = max(X, Y) + 1$
   - $U_2 = max(X, Z) + 1$
   - $U_3 = max(Y, Z) + 1$
   - $I = \lfloor \frac{L_1 + U_1}{2} \rfloor$
   - $J = \lfloor \frac{L_2 + U_2}{2} \rfloor$
   - $K = \lfloor \frac{L_3 + U_3}{2} \rfloor$

   Considere también la siguiente declaración:

   $$M \ : \ array \ [L_1..U_1] \ of \ array \ [L_2..U_2] \ of \ array \ [L_3..U_3] \ of \ T$$

   Suponiendo que M inicia en la dirección cero (0) y que el tamaño del tipo T es cuatro (4), se desea que calcule:

   (a) La dirección de $M[I][J][K]$ si las matrices se guardan en *row–major*.

   (b) La dirección de $M[I][J][K]$ si las matrices se guardan en *column–major*.

3. 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 viariantes *sin empaquetar*.
      - El lenguaje guarda registros y registros viariantes *empaquetados*.
      - El lenguaje guarda registros y registros viariantes *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 bifuración) mayor al 80%.

4. Considere un programa que suma todos los elementos de una matriz de dos dimensiones.

   Se desea que implemente, en el lenguaje C, dos versiones de este programa:

   - Recorre primero por fila y luego por columna.

     ```
     para cada i entre 0 y N-1
         para cada j entre 0 y M-1
             sumar arreglo[i,j]
     ```

   - Recorre primero por columna y luego por fila.

     ```
     para cada j entre 0 y M-1
         para cada i entre 0 y N-1
             sumar arreglo[i,j]
     ```

   Se desea que realice una medición de los tiempos de ejecución, ejecutando cada programa con matrices de tamaño $N \times M$, donde tanto N como M pueden tener los valores: 10^2, 10^3, 10^4, 10^5 y 10^6.

   Cada combinación debe ejecutarse al menos tres (3) veces.

   Para aquellas combinaciones que no caben en memoria, reportar este hecho y no seguir con el experimento.

   Realice un análisis de los tiempos obtenidos y de un razonamiento sobre el comportamiento observado.

   Como parte de su análisis, debe incluir respuestas a las siguientes preguntas (incluyendo sus justificaciones):

   - ¿Hay alguna diferencia en tiempo de ejecución entre las dos implementaciones propuestas?
   - ¿La forma de la matriz tiene algún efecto sobre el tiempo de la ejecución?
   - ¿Los tiempos de ejecución cambian al ejecutar más de una vez la misma configuración?
   - ¿Afecta a los tiempos de ejecución si la matriz se declara de forma global (memoria estática) o local (pila)?

   Para la elaboración de gráficas que muestren claramente sus resultados, es recomendable que se apoye en herramientas de visualización de datos (como los *plots* de Matlab, R, Octave, Excel, etc.)

   Nota: No hace falta que ingrese valores o inicialice la matriz de ninguna forma. El resultado como tal de la suma no es tan importante como el análisis de tiempo de ejecución.

5. **RETO EXTRA: *¡VELOCISTA!***

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

$$maldad(n) = trib\left(\left\lfloor \log_2(N_{n,\lfloor \log_2 n \rfloor})\right\rfloor + 1\right)$$

Queremos un programa que permita calcular valores para $maldad(n)$ de la forma más eficiente posible, para valores lo más grandes posibles de n.

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

- Reciba por la entrada estándar o argumento del sistema un valor para n, tal que $n \geq 2$ (esto puede suponerlo, no tiene que comprobarlo).
- Imprima el valor de $maldad(n)$.

Su programa debe imprimir el valor correcto y será probado con valores más y más grandes de n (no necesariamente acotado por la precisión disponible para un numero entero de 32 o 64 bits).

Reglas del reto: Intente desarrollar su programa de tal forma que pueda manejar valores grandes de n y responder eficientemente. Se impondrá un *timeout* de 1 segundo para cada invocación. Ganará quien pueda imprimir la respuesta correcta para el n más grande posible en ese tiempo.

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

Para evitar que se listen los valores de maldad(n), su programa debe pesar a lo sumo 32KB (en otras palabras, 2^{15} bytes).