Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información.
CI–4722 – Lenguajes de Programación III
Septiembre – Diciembre 2021

Examen I

(20 puntos)

A continuación encontrará 6 preguntas, cada una compuesta de diferentes sub–preguntas. El valor de cada pregunta (y sub–pregunta) estará expresado entre paréntesis al inicio de las mismas.

En aquellas preguntas donde se le pida ejecutar un algoritmo o procesar una entrada, incluya los pasos relevantes de la ejecución del mismo con los cuales usted pudo alcanzar su conclusión. Sea lo más detallado y preciso posible en sus razonamientos y procedimientos.

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 05 de Diciembre de 2021.

1. (2 puntos) Considere el siguiente fragmento de código, escrito en TAC:

   ```
   a := b + c
   b := a / 2
   b := d - c
   a := a + d
   c := b * a
   ```

   Al final de esta instrucción, las variables vivas son: $\{b, d\}$.

   Para cada instrucción, establezca el conjunto de variables vivas y la información de uso futuro para cada cada una de éstas.

2. (5 puntos) En clase, vimos como calcular el número de *Ershov* para expresiones de uno ($\otimes\ a$) o dos operandos ($a \otimes b$). Considere ahora expresiones de la siguiente forma: $\otimes(a, b, c)$ y código de cuatro direcciones FAC (*Four Address Code*), con instrucciones de la forma $\otimes\ R_r\ R_0\ R_1\ R_2$ que corresponde a $R_r := \otimes(R_0, R_1, R_2)$.

   a) (1 puntos) ¿Cómo se calcula $label(n)$, cuando n tiene tres hijos: c_{left}, c_{middle} y c_{right}?

   b) (1 puntos) Describa el proceso de generación de código usando este etiquetado extendido.

   c) (1 punto) Use el proceso diseñado en la pregunta anterior para generar código FAC de la siguiente expresión de alto nivel:

      ```
      (a ? b : c) + d * e
      ```

      Puede suponer que tiene registros ilimitados.

   d) (2 puntos) Para la misma expresión de la pregunta anterior, calcule los costos de evaluación contigua usando el algoritmo de programación dinámica. Para esto, suponga que tiene a los sumo tres (3) registros disponibles.

3. (2 puntos) Considere la siguiente instrucción, escrita en en el lenguaje de alto nivel a compilar:

```
we[have] = to[go[deep] + er]
```

a) (0.5 puntos) Traduzca a TAC, suponiendo que los símbolos `we`, `to` y `go` corresponden a la dirección del primer elemento de arreglos de enteros.

b) (1.5 puntos) Genere código de máquina para el fragmento empleando el algoritmo basado en Asignación de Registros según Descriptores de Uso y Asignación. Suponga que dispone de tres (3) registros para el cómputo de las expresiones. Muestre los descriptores en cada paso de generación.

4. (3 puntos) Considere el siguiente fragmento de código en pseudo–código que realiza la operación *merge*:

```
i := 0;
j := 0;
k := 0;
while (i < n && j < m) {
    if (a[i] < b[j]) {
        c[k] := a[i];
        i := i + 1;
    } else {
        c[k] := b[j];
        j := j + 1;
    }
    k := k + 1;
}
while (i < n) {
    c[k] := a[i];
    i := i + 1;
    k := k + 1;
}
while (j < m) {
    c[k] := b[j];
    j := j + 1;
    k := k + 1;
}
```

Puede suponer que n y m son enteros positivos. También puede suponer que a es un arreglo de enteros de tamaño n, b es un arreglo de enteros de tamaño m y c es un arreglo de enteros de tamaño $n + m$.

a) (1 punto) Construya el TAC que se genera a partir del fragmento de código anterior.

b) (0.5 puntos) Del TAC obtenido, construya el grafo de flujo asociado.

c) (1.5 puntos) Del TAC obtenido, construya el árbol de dominadores, mostrando los diferentes tipos de aristas resultantes (avance, retroceso, retorno y cruce). Con esta información identifique los ciclos naturales y diga si el grafo es reducible o no.

5. (3 puntos) Sea un número entero positivo n. Se construye un nuevo número n' de la siguiente forma:

   - Si n es par, entonces $n' = \frac{n}{2}$.
   - Si n es impar, entonces $n' = 3 \times n + 1$

   La conjetura de Collatz establece que si se vuelve a aplicar este proceso de forma repetida, tomando n' como entrada al mismo proceso, eventualmente se alcanza el número 1.

   a) (0.5 puntos) Escriba un programa en pseudo–código que, dado un n, devuelva la cantidad de pasos necesarios para alcanzar 1 siguiendo el procedimiento descrito anteriormente.

   b) (0.5 puntos) Traduzca su programa a TAC.

   c) (0.5 puntos) Construya el grafo de flujo para el programa en TAC generado.

   d) (1.5 puntos) Construya el grafo de interferencia y genere código con Asignación de Registros por Coloración de Grafos, suponiendo que se dispone de dos (2) registros.

6. (5 puntos) Considere un TAC cuyas expresiones son todas números naturales (esto es, no permiten números negativos) y todas las variables son inicializadas en un número natural arbitrario.

   La sintaxis y operaciones disponibles para este TAC se entienden con la siguiente gramática:

$$
\begin{array}{rcl}
TAC & \rightarrow & TAC \quad \backslash n \quad LINE \\
 & | & LINE \\
LINE & \rightarrow & \textbf{id} \quad : \quad INSTR \\
 & | & INSTR \\
INSTR & \rightarrow & \textbf{id} \quad := \quad ARIT \\
 & | & advance \quad \textbf{id} \\
 & | & goto \quad \textbf{id} \\
 & | & goif \quad CMP \quad \textbf{id} \\
ARIT & \rightarrow & \textbf{num} \quad + \quad \textbf{num} \\
 & | & \textbf{num} \quad - \quad \textbf{num} \\
 & | & \textbf{num} \quad * \quad \textbf{num} \\
 & | & \textbf{num} \quad / \quad \textbf{num} \\
CMP & \rightarrow & \textbf{num} \quad < \quad \textbf{num} \\
 & | & \textbf{num} \quad > \quad \textbf{num} \\
 & | & \textbf{num} \quad == \quad \textbf{num} \\
 & | & \textbf{num} \quad != \quad \textbf{num}
\end{array}
$$

   La mayoría de las operaciones aritméticas y relacionales tienen la semántica convencional. Además, existirá una instrucción especial `advance`, de forma tal que `advance x` es equivalente a `x = x + 1`. Las operaciones de resta y división, sin embargo, tienen semánticas especiales en este contexto:

   - La resta es siempre mayor o igual a cero: $a -_{\mathbb{N}} b = max(a -_{\mathbb{Z}} b, 0)$
   - La división es truncada al natural inmediatamente inferior: $a /_{\mathbb{N}} b = \lfloor a /_{\mathbb{R}} b \rfloor$

Considerando este TAC que trabaja sobre números naturales:

a) (1 punto) Queremos saber si una variable en nuestro TAC puede llegar a ser igual a cero (por algún camino posible de ejecución).

Como ejemplo, consideremos el siguiente programa:

```
      advance b
      goif b < c then
      a := b - c
      goto end
then: a := b + c
end:  c := b / c
```

Notemos que dependiendo de los valores iniciales (que son desconocidos) el flujo puede seguir uno de dos caminos hasta llegar a la última instrucción.

Primeramente se avanza `b`, haciendo que sea imposible que `b` sea igual a cero. Luego, si `b < c` se hace la asignación `a := b + c`. Después de esta instrucción es imposible que `a` sea cero, ya que `b` no puede ser cero y sumarle un número natural cualquiera no puede producir cero. Si, por el contrario, `b >= c` entonces se ejecuta la asignación `a := b - c`. Independientemente del valor de `b`, una resta siempre podrá resultar en cero. Finalmente, la última instrucción es una asignación `c := b / c`, por lo que es posible que `c` siga valiendo cero (también es posible que esté indefinida, pero aún no nos preocuparemos por eso).

En conclusión, justo después de la última instrucción, es posible que `a` y `c` sean cero, pero imposible que `b` lo sea.

Plantee el problema de encontrar las variables que son potencialmente cero para cada instrucción como un problema de flujo de datos, proponiendo la construcción de IN y OUT (incluyendo el estado inicial) y la función de transferencia asociada. Diga también si dicho razonamiento será *hacia adelante* o *hacia atrás*.

b) (1 punto) Usando la información de qué variables son potencialmente cero, establezca cuáles expresiones de división tienen riesgo de estar indefinidas (por tener denominador igual a cero).

Considerando el mismo ejemplo de la parte anterior, la asignación `c := b / c` es de riesgo, pues es posible que `c` haya sido igual a cero antes de evaluar `b / c`.

c) (3 puntos) Implemente un parser recursivo descendente para el TAC utilizado en esta pregunta. Luego, construya el grafo de flujo para el mismo, aplique el análisis de flujo propuesto por usted en las dos partes anteriores y reporte como *warnings* las expresiones de división que potencialmente están indefinidas.

Deberá reportar también errores léxicos, sintácticos y de etiquetas indefinidas. Recuerde que todas las variables están definidas con algún número natural arbitrario al momento de iniciar la ejecución de un programa.