Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información.
CI–4721 – Lenguajes de Programación II
Enero – Marzo 2025

Tarea IV

(10 puntos)

A continuación encontrará 3 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.

La entrega se realizará por correo electrónico a rmonascal@gmail.com hasta las 11:59pm. VET del Miércoles 2 de Abril de 2025.

1. (4 puntos) Generación de código de tres direcciones con corto circuito.

   a) (1 punto) Considere la gramática de expresiones booleanas vista en clase y auméntela con la producción:

   $$B \quad \rightarrow \quad B \Rightarrow B$$

   Esta producción representa expresiones de implicación. Esto es, si se tiene $a \Rightarrow b$, la expresión evalúa a *true* si el resultado de evaluar a implica lógicamente al resultado de evaluar `b`.

   Construya la regla asociada a esta producción que utiliza `B.true` (etiqueta a la cual debe irse en caso de evaluar en `true`) y `B.false` (etiqueta a la cual debe irse en caso de evaluar en `false`), y calcula `B.code` (código generado para evaluar la expresión). Debe hacer uso de la técnica de *jumping code*.

   b) (1 punto) Para la producción anterior, construir su implementación usando *backpatching*; esto es, con atributos `B.truelist` y `B.falselist`. Recuerde que cuenta con las funciones *makelist*, *backpatch*, *merge* y *gen*. Para esta pregunta, puede ignorar el atributo `B.code`, ya que quedan implícito mediante el uso de *gen*. Puede agregar símbolos M_i en las producciones que crea conveniente (con producciones de tipo $M_i \rightarrow \lambda$), para guardar la dirección de la instrucción actual.

   c) (1 punto) Considere la gramática de instrucciones vista en clase y auméntela con la producción:

   $$S \quad \rightarrow \quad \textit{while } B_1 : S_1 \ \& \ B_2 : S_2$$

   Esta producción representa un tipo de iteración indeterminada. Esto es, si se tiene $\textit{while } a : b \ \& \ c : d$, se ejecuta la instrucción mientras alguno entre B_1 y B_2 evalúe a `true`. El comenzar cada iteración, si B_1 evalúa a `true` entonces se ejecuta S_1. De lo contrario, si B_2 evalúa a `true`, entonces se ejecuta S_2. De lo contrario, la ejecución del ciclo es interrumpida.

   Construya la regla asociada a esta producción que utiliza `S.next` (etiqueta a la cual debe irse en caso de finalizar la ejecución de `S`) y calcula `S.code` (código generado para ejecutar la instrucción). Debe hacer uso de la técnica de *jumping code*.

   d) (1 punto) Para la producción anterior, construir su implementación usando *backpatching*; esto es, con atributos `B.truelist`, `B.falselist` y `S.nextlist`. Recuerde que cuenta con las funciones *makelist*, *backpatch*, *merge* y *gen*. Para esta pregunta, puede ignorar los atributos `B.code` y `S.code`, ya que quedan implícitos mediante el uso de *gen*. Puede agregar símbolos M_i en las producciones que crea conveniente (con producciones de tipo $M_i \rightarrow \lambda$), para guardar la dirección de la instrucción actual.

2. (4 puntos) Considere el siguiente fragmento de código:

```
s, i = 0, 0;
while (i < 10) do
   s, i = s + (i * i) / 2, i + 1;
```

   a) (2 puntos) Escriba el código de máquina de pila que sería generado por dicho código, con el método visto en clase.

   b) (2 puntos) Escriba el código de máquina de tres direcciones que sería generado por dicho código, con el método visto en clase.

3. (2 puntos) En su más reciente publicación, Ehlbrin K. Pozzo, *et al.* hicieron mejoras sobre el proceso de generación de código usando *JumpingCode*.

   Se desea que lea dicho paper y realice una implementación de lo que allí se propone.

   Puede encontrar dicho paper dando clic en el enlace siguiente: Pozzo et al. - On Taking Jumping Code to New Computational Heights