Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información.
CI–4721 – Lenguajes de Programación II
Abril – Julio 2021

Examen II

(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 Miércoles 28 de Julio de 2021.

1. (2 puntos) Considere las siguientes declaraciones de tipo de datos:

```
type pokemon = struct
   pi : float
   ka : pokebola
   chu : pokebola
end

type pokebola = *pokemon
```

   a) (1 punto) Construya el grafo de tipos que se genera para representar al tipo `pokemon`, considerando equivalencia por nombre estricta.

   b) (1 punto) Considere el siguiente conjunto de declaraciones:

```
var a : pokemon
var b = &a : Tb
var c = (*b).chu : Tc

var d = (*a.chu).pi
var e = *(*(*c).ka).chu
```

   Para las variables `a`, `b`, `c`, `d` y `e`, establezca su tipo (note que para la variable `a` ya está colocado) y marque, en el grafo de tipos de la pregunta anterior, a qué nodo corresponde el tipo de cada una de las variables mencionadas.

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

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

   a) (1 punto) Escriba el código de máquina de pila que sería generado por dicho código, con el método visto en clase.

   b) (1 punto) Escriba el código de máquina de tres direcciones que sería generado por dicho código, con el método visto en clase.

3. (5 puntos) Traducción dirigida por sintaxis para verificación de tipos.

   a) (2 puntos) Considere la siguiente gramática de expresiones:

$$
\begin{array}{rcl}
E & \rightarrow & E + E \\
 & | & E \wedge E \\
 & | & E < E \\
 & | & E \ ?: E \\
 & | & E \ !! \\
 & | & (E) \\
 & | & num \\
 & | & true \\
 & | & false \\
 & | & null
\end{array}
$$

   Las primeras tres reglas de la producción corresponden a la suma de enteros, la conjunción booleana y el operador relacional "menor que", respectivamente.

   La producción $E \rightarrow E \ ?: E$ representa evaluaciones con valores por defecto en caso de nulidad. Esto es, si se tiene $a \ ?: b$, la expresión evalúa a a si $a = null$ y evalúa a b de lo contrario.

   La producción $E \rightarrow E \ !!$ representa evaluaciones forzadas sin seguridad por nulidad. Esto es, si se tiene $a \ !!$, la expresión evalúa a a si $a \neq null$ y es un error de tipo de lo contrario.

   Construya la regla asociada a esta producción que calcula `E.type`, el tipo de la expresión. El atributo debe ser sintetizado.

   Nota: Puede suponer que el tipo de num es INT, el tipo de $true$ y $false$ es $BOOL$ y el tipo de $null$ es $NULL$.

   b) (1 punto) Utilice el esquema de traducción dirigido por sintaxis definido en la pregunta anterior para construir el tipo de la siguiente frase:

$$(((null \ ?: 42) + (69 \ !!) < (7 \ ? : null)) \wedge null) \ ?: true$$

   Muestre cada uno de los pasos en el árbol de derivación.

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

$$S \rightarrow repeatWhen \ E \ lt \ S \ gt \ S$$

   Esta producción representa repeticiones indeterminadas, con cuerpos que dependen del signo de la una expresión aritmética. Esto es, si se tiene `repeatWhen a lt b gt c`, la expresión ejecuta `b` siempre que `a <0` y ejecuta `c` siempre que `a >0`. En el caso que `a = 0` entonces sale de la repetición.

   Construya la regla asociada a esta producción que calcula `S.type`, el tipo de la instrucción (recuerde que las instrucciones son siempre de tipo `void` a menos que haya ocurrido algún error). El atributo debe ser sintetizado.

   d) (1 punto) Replantee las reglas definidas para las gramáticas anteriores en términos de reglas de inferencia.

4. (2 puntos) Considere los siguiente símbolos con su tipos (potencialmente polimórficos):

`cmap`	:	β	`head`	:	$\forall\alpha.list(\alpha) \to \alpha$
`f`	:	γ	`tail`	:	$\forall\alpha.list(\alpha) \to list(\alpha)$
`x`	:	ρ	`if`	:	$\forall\alpha.bool \times \alpha \times \alpha \to \alpha$
`[]`	:	$\forall\alpha.list(\alpha)$	`concat`	:	$\forall\alpha.list(\alpha) \to list(\alpha) \to list(\alpha)$
`null`	:	$\forall\alpha.list(\alpha) \to bool$	`match`	:	$\forall\alpha.\alpha \times \alpha \to \alpha$

Considere también la siguiente expresión:

```
match(cmap(f,x), if(null(x), [], concat(f(head(x)), cmap(f, tail(x)))))
```

Utilice el esquema de verificación visto en clase para determinar el tipo más general de `foldl` y muestre las unificaciones realizadas en cada paso.

5. (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 una producción de la forma:

   $$B \quad \to \quad B \Rightarrow B$$

   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 $M_i \to \lambda$), para guardar la dirección de la instrucción actual.

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

   $$S \quad \to \quad while\ B_1 : S_1\ \&\ B_2 : S_2$$

   Esta producción representa un tipo de iteración indeterminada. Esto es, si se tiene $while\ a : b\ \&\ c : d$, 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 $M_i \to \lambda$), para guardar la dirección de la instrucción actual.

6. (5 puntos) Se desea que modele e implemente, en el lenguaje de su elección, un intérprete para una máquina de pila.

   Su intérprete debe saber procesar/entender las siguientes instrucciones:

   a) Manipulación directa de la pila:

      1) `PUSH <val>`

         Empila el valor contenido en `<val>`. Ese valor únicamente debe ser un número entero o un literal boolean (`true` o `false`)

      2) `POP`

         Desempila y descarta lo que se encuentre en el tope de la pila.

   b) Operaciones aritméticas, booleanas, relaciones y de igualdad:

      1) `ADD`, `SUB`, `MUL`, `DIV`

         Desempila los dos últimos valores de la pila, realiza la operación aritmética correspondiente sobre ellos y empila el resultado.

         El programa debe reportar un error e ignorar la acción si los valor desempilados no son enteros o si la pila no tiene suficientes elementos.

      2) `AND`, `OR`

         Desempila los dos últimos valores de la pila, realiza la operación booleana correspondiente sobre ellos y empila el resultado.

         El programa debe reportar un error e ignorar la acción si los valor desempilados no son booleanos o si la pila no tiene suficientes elementos.

      3) `LT`, `LE`, `GT`, `GE`

         Desempila los dos últimos valores de la pila, realiza la comparación relacional correspondiente sobre ellos y empila el resultado.

         El programa debe reportar un error e ignorar la acción si los valor desempilados no son enteros o si la pila no tiene suficientes elementos.

      4) `EQ`, `NEQ`

         Desempila los dos últimos valores de la pila, realiza la comparación de igualdad correspondiente sobre ellos y empila el resultado.

         El programa debe reportar un error e ignorar la acción si los valor desempilados no son del mismo tipo o si la pila no tiene suficientes elementos.

      5) `UMINUS`

         Desempila el último valor de la pila, realiza un negativo aritmético sobe el mismo y empila el resultado.

         El programa debe reportar un error e ignorar la acción si el valor desempilado no es entero o si la pila está vacía.

      6) `NOT`

         Desempila el último valor de la pila, realiza una negación booleana sobe el mismo y empila el resultado.

         El programa debe reportar un error e ignorar la acción si el valor desempilado no es booleano o si la pila está vacía.

c) Manejo de identificadores y asignaciones:

1) `RVALUE <id>`

   Empila el *contenido* del identificador `<id>`.

   El programa debe reportar un error e ignorar la acción si el identificador aún no tiene un valor asignado.

2) `LVALUE <id>`

   Empila la *dirección* del identificador `<id>`.

   Nótese que esta instrucción es siempre válida, pues todos los identificadores existen por defecto, aún si no se les ha asignado un valor concreto.

3) `ASSIGN`

   Desempila los últimos dos elementos de la pila. Asigna en la dirección representada por el elemento que estaba en el tope el valor del elemento que estaba debajo.

   El programa debe reportar un error e ignorar la acción si el valor desempilado en el tope no corresponde con un l–value o si la pila no tiene suficientes elementos.

d) Saltos condicionales e incondicionales:

1) `GOTO <et>`

   Salta a la instrucción con etiqueta `et`.

   El programa debe reportar un error e ignorar la acción si la etiqueta no está asociada a ninguna instrucción.

2) `GOTRUE <et>`

   Salta a la instrucción con etiqueta `et` solamente si el valor en el tope de la pila es la constante booleana `true`.

   El programa debe reportar un error e ignorar la acción si la etiqueta no está asociada a ninguna instrucción, si el valor en el tope de la pila no es booleano o si la pila está vacía.

3) `GOFALSE <et>`

   Salta a la instrucción con etiqueta `et` solamente si el valor en el tope de la pila es la constante booleana `false`.

   El programa debe reportar un error e ignorar la acción si la etiqueta no está asociada a ninguna instrucción, si el valor en el tope de la pila no es booleano o si la pila está vacía.

e) Instrucciones de entrada/salida:

1) `READ <id>`

   Solicita al usuario un nuevo valor y lo coloca como valor del identificador `<id>`.

   El programa debe reportar un error e ignorar la acción si el valor proporcionando por el usuario no es un literal entero o booleano.

2) `PRINT <id>`

   Imprime el valor almacenado en el identificador `<id>`.

   El programa debe reportar un error e ignorar la acción el identificador no tiene valor alguno asociado.

f) Meta–instrucciones de control:

1) `RESET`

   Regresa el simulador a su estado inicial, vaciando la pila, desasociando todos los identificadores con los valores que almacenaban así como todas las etiquetas de las instrucciones a las que referían.

2) `EXIT`

   Debe salir del simulador.

Todas las instrucciones pueden estar opcionalmente etiquetadas, con la siguiente sintaxis: `<et>: <instr>`, donde `<et>` es una etiqueta y `<instr>` una instrucción.

Por ejemplo:

- `deepThought: PUSH 42`
- `TH: OR`
- `GOTO: GOTO GOTO`

Las etiquetas e identificadores pueden estar formadados por cualquier cadena de caracteres no–vacía, tanto en minúscula como en mayúscula (incluso si hay choques con instrucciones)

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 %.