UNIVERSIDAD TECNOLÓGICA CENTROAMERICANA

Ingeniería en Sistemas Computacionales
COMPILADORES I
REPOSICIÓN (Valor 30%)

NOTA

Nombre del estudiante:_____	**No. Cuenta:**_____
Docente: Carlos Varela	
Fecha de Examen : 22/03/2022	**Fecha y firma de Revisión:** _____

Instrucciones generales:

- Deberá mantenerse conectado en la sesión de **Blackboard Collaborate** hasta el final de la hora clase (6:40 P.M).
- El espacio para subir su examen vence a las 11:59 P.M., debe subir su examen antes de dicha hora.
- El plagio, copia parcial o completa en su examen será penalizado con 0%.

DotNetWeb es un framework para desarrollo web basado en plantillas y estructuras simples de código en c#. El framework combina código basado en c# y lenguaje de marcado (HTML o XML) en una sola plantilla que al compilarla genera un HTML o XML valido. La estructura de la plantilla es la siguiente:

- Init: esta sección siempre va al inicio del archivo a compilar, puede contener sentencias de declaración y asignación de variables. Las variables declaradas aquí existirán en toda la plantilla definida. Se aceptan los tipos: int, float, string y arreglos de estos mismos tipos. Esta sección se ve algo así: "{% init **code here** %}".
- HTML: después de la sección init se debe escribir por lo menos un nodo HTML, solo se aceptan nodos que utilicen una etiqueta para abrir y una etiqueta para cerrar. Ejemplo: . No se aceptan nodos que cierren en la misma etiqueta. Ejemplo:
- Sentencias: dentro de cada nodo HTML se acepta el uso de las sentencias if, foreach y string interpolation.
    - La sentencia if evalúa una condición y si se cumple muestra lo que está dentro del if. Sintaxis: "{% if %} **html_here** {% endif %}
    - La sentencia foreach permite iterar un arreglo y repite todo lo que esté dentro del foreach. Sintaxis: {-%foreach element in arr %} **html_here** { %endforeach %}
    - El string interpolation va a remplazar la expresión entre llaves "{{ }}" por el resultado de evaluar esa expresión.

UNIVERSIDAD TECNOLÓGICA CENTROAMERICANA

Requerimientos:

1. Desarrollar el analizador sintáctico (parser). En la última página de este documento se provee una gramática parcial para este lenguaje. A la gramática le hace falta las producciones de sentencias, debe agregar estas producciones y luego construir su parser. **(20%)**
2. Realizar la validación semántica **(25%)**
    a. Validaciones de scope. Solo existe un scope en toda la plantilla. Se debe validar que no se redefina una variable que ya existe.
    b. Validaciones de tipo para las expresiones. Validar que las operaciones sean validas entre los tipos utilizados.
        i. Todas las operaciones aritméticas entre números enteros o números flotantes y combinar enteros con flotantes. (Al realizar una operación aritmética entre entero y flotante el resultado es un flotante).
        ii. La operación de suma entre string y string o string y cualquier otro tipo.
        iii. Se pueden comparar enteros con enteros, flotantes con flotantes, flotantes con enteros y strings con strings.
3. Interpretación de código: Se debe interpretar el código de la plantilla, de manera que no existan expresiones ni sentencias, solo el resultado de evaluar esas expresiones y sentencias. **(35%)**
4. Generación de código: la salida de su compilador debe ser un archivo HTML que contenga el código interpretado y represente un HTML valido. Este archivo debe poder abrirse en un navegador y observar el resultado del código interpretado. **(20%)**

Ejemplo de una plantilla:

```
{% init
    int a;
    float b;
    string x;
    StringList arr;
    string element;

    a = 5;
    b = 5.68;
    x = 'test';
    arr = ['one', 'two', 'three'];
%}

<div>
    {{a}}
</div>

<div>
{%if a > 0 %} <b>{{ b }}</b>{% endif %}
</div>
<ul>
{-%foreach element in arr %}
    <li> {{element}} </li>
{ %endforeach %}
</ul>
```

UNIVERSIDAD TECNOLÓGICA CENTROAMERICANA

Gramática:

S → init template
init → {% 'init' code %}
code → decls assignations
| ε
decls → decl decls
| decl
decl → type id ;
assignations → assignation assignations
| ε
assignation → id = eq
template → tag template
| tag
tag → < id > stmts </ id >

eq → rel eq'
eq' → == rel eq'
| <> rel eq'
| ε
rel → expr < expr
| expr > expr
| expr
expr → term expr'
expr' → + term expr'
| - term expr'
| ε
term → factor term'
term' → * factor term'
| / factor term'
| ε
factor → digit
| id
| [expr-list]
| (eq)
expr-list → eq , expr-list
| eq