Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Estado de México

Escuela de Ingeniería y Ciencias, Región Ciudad de México

Departamento de Computación

Examen final de práctica de Diseño de compiladores

Profesor: Ariel Ortiz Ramírez

INSTRUCCIONES

Debes resolver el siguiente problema usando tu propia computadora portátil. Puedes usar cualquier información y/o material contenido en tu computadora, así como cualquier material escrito (manuales, listados de programa, libros, notas, etc.). Durante el examen no se permite hablar o compartir materiales con alguien más.

La solución completa del problema debe quedar en un solo archivo fuente llamado `A0MMMMMM.cs` (donde `A0MMMMMM` es tu matrícula). Entrega este archivo a tu profesor. Asegúrate que el archivo fuente incluya en la parte superior tu información personal (nombre y matrícula) dentro de un comentario.

Tiempo límite: 120 minutos.

DESCRIPCIÓN DEL PROBLEMA

Implementa el lenguaje de programación PPT (Piedra, Papel y Tijeras) que se describe a continuación:

- Un programa en este lenguaje consiste de una sola expresión, la cual está compuesta de operadores y operandos.
- Los operandos pueden ser una de las siguientes cadenas de caracteres: `piedra`, `papel` y `tijeras`.
- Los operadores soportados en el lenguaje son: + (más) y − (menos). Ambos operadores son binarios e infijos. El operador de + tiene mayor precedencia que el operador de −. El operador + tiene asociatividad izquierda, mientras que el operador − tiene asociatividad derecha. La semántica de los operadores es la siguiente:

$exp_1 + exp_2$	Devuelve el valor que gana entre sus dos operandos: `piedra` le gana a las `tijeras`, las `tijeras` le gana al `papel`, y el `papel` le gana a la `piedra`. Si ambos operandos son iguales, devuelve el valor único.
$exp_1 - exp_2$	Devuelve el valor que pierde entre sus dos operandos: `piedra` pierde ante `papel`, el `papel` pierde ante las `tijeras`, y las `tijeras` pierden ante la `piedra`. Si ambos operandos son iguales, devuelve el valor único.

  La funcionalidad de los operadores + y − está provista en el archivo `execute.js` como funciones externas escritas en JavaScript y disponibles en el código textual de WebAssembly como `$mas` y `$menos`, respectivamente.

- Se pueden utilizar paréntesis para agrupar sub-expresiones y lograr un orden de evaluación diferente al impuesto por las reglas de precedencia y asociatividad descritas anteriormente.
- Espacios y tabuladores deben ser ignorados.

Ejemplos:

Programa Fuente	*Salida del Programa Ejecutable*
`piedra + papel`	`papel`
`papel - tijeras`	`papel`
`(piedra - papel) - tijeras + piedra - papel`	`piedra`
`piedra + papel + tijeras - piedra - papel`	`tijeras`

Escribe un programa en C# que reciba una expresión del lenguaje PPT y realice el análisis sintáctico, construcción del árbol AST, y generación de código textual de WebAssembly. Considera lo siguiente:

- La expresión de entrada se debe tomar como argumento de la línea de comando.
- Si hay un error sintáctico en la entrada, se debe desplegar el mensaje “parse error”, y el programa debe terminar.
- Si no se detectan errores sintácticos, se debe desplegar el AST.
- La salida de código textual en WebAssembly debe ir siempre a un archivo llamado `output.wat`.
- El propósito del código generado es evaluar a tiempo de ejecución la expresión de entrada y devolver el resultado.
- El comando `execute.js` será invocado manualmente desde la terminal.

UN EJEMPLO COMPLETO

Al ejecutar el siguiente comando desde la terminal:

```
mono ppt.exe 'piedra + papel + tijeras - piedra - papel'
```

se debe desplegar el siguiente AST en la salida estándar:

```
Programa
  Menos
    Mas
      Mas
        Piedra
        Papel
      Tijeras
    Menos
      Piedra
      Papel
```

y el contenido del archivo `output.wat` debe ser:

```
(module
  (import "ppt" "mas" (func $mas (param i32) (param i32) (result i32)))
  (import "ppt" "menos" (func $menos (param i32) (param i32) (result i32)))
  (func
    (export "inicio")
    (result i32)
    i32.const 0 ;; Piedra
    i32.const 1 ;; Papel
    call $mas
    i32.const 2 ;; Tijeras
    call $mas
    i32.const 0 ;; Piedra
    i32.const 1 ;; Papel
    call $menos
    call $menos
  )
)
```

El *script* `execute.js` proporcionado se encarga de compilar y ejecutar el código de WebAssembly. En la terminal, este comando:

```
node execute.js
```

debe producir la siguiente salida:

```
tijeras
```

EVALUACIÓN

Las ponderaciones son las siguientes:

1. **10** El código de C# compila sin errores.
2. **30** Cumple el punto 1, adicionalmente: realiza análisis léxico y sintáctico sin errores.
3. **50** Cumple los puntos 1 y 2, adicionalmente: construye e imprime el AST sin errores.
4. **100** Cumple los puntos 1, 2 y 3, adicionalmente: genera código de ensamblador CIL sin errores.

"Computer language design is just like a stroll in the park. Jurassic Park, that is."

– Larry Wall, creator of Perl.