Quinto boletín de prácticas sobre MMPor

IG29: Compiladores e intérpretes

Sesiones séptima y octava de prácticas

1. Introducción

El objetivo final de esta práctica será introducir en tu calculadora MMPor una estructura de control condicional. Dado el estado actual de la calculadora, sin ningún medio para expresar condiciones lógicas (es decir, que puedan evaluarse para decidir si son "verdaderas" o "falsas"), antes de introducir la estructura de control propiamente dicha será necesario ampliar el lenguaje de expresiones de MMPor.

2. Ampliación del lenguaje de expresiones

Hasta este punto, los únicos operadores de MMPor eran suma, resta y multiplicación, todos ellos binarios infijos y asociativos por la izquierda. Además, suma y resta eran igual de prioritarias y la multiplicación tenía mayor prioridad. Se trata, por una parte, de añadir a los anteriores operadores, aritméticos, otros que faciliten la expresión de condiciones: de comparación y lógicos (negación `!`, conjunción `&&` y disyunción `||`); por otra, completaremos los operadores aritméticos con los signos + y − como unarios prefijos. Los niveles de prioridad resultantes serán los siguientes, de mayor a menor:

1. Unarios prefijos: `+ - !`
2. Multiplicación: `*`
3. Suma y resta: `+ -`
4. Comparadores: `< <= > >= == !=`
5. Conjunción: `&&`
6. Disyunción: `||`

Salvo los operadores unarios, todos serán binarios infijos y asociativos por la izquierda. Además, aunque de momento esto no tenga casi ninguna repercusión en el comportamiento observable de la calculadora, la evaluación de todos los operadores debe seguir las reglas siguientes:

- Cada operando debe evaluarse, a lo sumo, una vez.
- El orden de evaluación de los operandos será de izquierda a derecha.
- Han de evaluarse todos los operandos excepto en el caso de los operadores lógicos binarios, `&&` y `||`, que se evaluarán por circuito corto, esto es, si el valor del operando izquierdo determina el resultado de la operación (es falso en el caso `&&` o es verdadero en el caso `||`), no se debe evaluar el derecho[1].

Dado que la semántica de las nuevas operaciones introducidas en MMPor es la habitual, para completar su especificación sólo queda abordar el asunto de la representación de los valores lógicos, en general y como resultado de diferentes operaciones. A este respecto, cabe decir lo siguiente:

- A MMPor no se le añade un tipo lógico explícito, sino que, de momento, seguirá manejando únicamente números enteros en sus expresiones.

1. Piensa qué programas MMPor podrían tener un comportamiento diferente si nunca se aplicara circuito corto; en particular, considera la posibilidad de que un programa pueda producir o no un error dependiendo del tipo de circuito, corto o largo, aplicado a la evaluación de conjunciones y disyunciones. Además, piensa qué podría suceder si a MMPor se le dotara de operadores con efectos secundarios, como son por ejemplo los incrementos y decrementos de C.

- Cuando sea necesario interpretar un valor entero como lógico (por ejemplo, como operando de una operación lógica), los estrictamente positivos se interpretarán como verdaderos y los demás (cero y negativos), como falsos.
- Cuando una operación deba devolver un valor lógico (ya sea una comparación o una operación lógica), un resultado verdadero se representará con el entero positivo $+1$; uno falso, con -1.

Para introducir en tu calculadora todos los cambios que se te piden en este apartado, puedes seguir, por ejemplo, estos pasos:

(1) Diseña las clases de nodos que necesitas para representar en tus ASTs las nuevas operaciones (los signos, las lógicas y las comparaciones). Piensa si necesitas o no representar semánticamente el operador + unario. Considera también la posibilidad de que varias operaciones similares sean representadas por una única clase de nodo con algún atributo que sirva para diferenciar una operación de otra; puedes probar a juntar Más y Menos en una única clase como experiencia antes de decidir qué haces con las nuevas operaciones.

(2) Implementa las nuevas clases de nodos en tu módulo `AST.py`. No olvides tener en cuenta las correspondientes reglas de evaluación ni tampoco que la opción `-s` debe seguir funcionando correctamente, respetando el formato de entrada de verArbol.

(3) Introduce las modificaciones necesarias en el nivel léxico: primero, en `disenyo.txt`; después, en `mmpor.mc`. Observa que se trata, únicamente, de introducir nuevos operadores.

(4) Modifica tu gramática en `disenyo.txt` para introducir los cuatro nuevos niveles de prioridad en las expresiones.

(5) Teniendo en cuenta cuál es tu nueva gramática y cuáles son las clases de nodos que has diseñado (e implementado) para tus ASTs, modifica adecuadamente tu esquema de traducción para que construya el correspondiente AST en cada caso; primero, modifica el esquema en `disenyo.txt` y, después, hazlo en `mmpor.mc`.

(6) Crea nuevos ficheros p*XX*`.mmp` y p*XX*`.sal` para probar las nuevas capacidades de tu calculadora. Verifica que ésta supera con éxito las nuevas pruebas y que tampoco falla con las anteriores. Comprueba también su funcionamiento con la opción `-s`.

3. Estructura de control condicional

Una vez has dotado a tu calculadora de la capacidad de aceptar expresiones que representan condiciones, ya puedes introducir diferentes estructuras de control en su lenguaje de entrada. Para empezar, en esta práctica vas a introducir una estructura condicional múltiple con la siguiente sintaxis:

- La estructura debe contar obligatoriamente con una línea de cabecera y una línea de cierre.
- La línea de cabecera consistirá en una condición (es decir, una expresión que se interpretará como verdadera o falsa) rodeada por las nuevas palabras reservadas `if` y `then`.
- La línea de cierre contendrá la nueva palabra reservada `end`.
- Tras la línea de cabecera debe aparecer la secuencia de sentencias que deben ejecutarse sólo si la condición resulta ser verdadera. La secuencia puede ser de cero o más elementos y cada uno de ellos podrá ser una asignación, una sentencia de escritura o, a su vez, una estructura condicional.
- Tras la anterior secuencia de sentencias, puede aparecer una secuencia de bloques `elif`, cada uno de los cuales debe componerse, a su vez, de una cabecera y una secuencia de cero o más sentencias. La cabecera de un bloque `elif` será una línea consistente en una condición rodeada por las palabras reservadas `elif` (nueva) y `then`; las sentencias del bloque sólo se ejecutarán si tanto la condición del `if` como la de los bloques `elif` anteriores resultaron ser falsas y, por contra, la de la cabecera del bloque actual resulta ser verdadera.
- Antes de la línea de cierre, puede aparecer un bloque `else`. De aparecer, se considerará completamente equivalente a un último bloque `elif` con

  ```
  elif 1 then
  ```

  en su cabecera, aunque la sintaxis de su línea de cabecera incluirá únicamente la nueva palabra reservada `else`.

Un ejemplo de estructura condicional podría ser éste:

```
if a<0 then
    print "Necesito, al menos, un signo negativo"
    SoloUnDig:=0
elif a>99 then
    print "Necesito varios dígitos"
    SoloUnDig:=0
elif a>=10 then
    print "Necesito exactamente dos dígitos"
    SoloUnDig:=0
else
    print "El valor de a se representa con sólo un dígito: ", a
    SoloUnDig:=1
end
```

En general, la estructura condicional será un nuevo tipo de sentencia (como la asignación o la sentencia de escritura) que, sintácticamente, podrá aparecer en los mismos contextos que las otras sentencias.

Como puedes suponer, se trata de que diseñes, implementes y pruebes los cambios necesarios para que tu calculadora admita la presencia de las correspondientes estructuras de control condicional en su lenguaje de entrada[2].

4. Entrega de la práctica

Como resultado de esta práctica, debes entregar en un paquete `entrega06.tgz` los ficheros siguientes:

- `disenyo.txt`.
- `mmpor.mc`, `AST.py`, `memo.py` y `errores.py`.
- Los correspondientes ficheros de prueba, incluyendo los nuevos.

2. Y, si quieres ir preparándote para el trabajo del próximo boletín, ve pensando en cómo introducirías en tu calculadora una estructura de control repetitiva.