Procesadores del Lenguaje

Curso 2024-2025

Práctica FINAL. Primera Aproximación.

Traductor de expresiones a notación postfija

Introducción

La práctica final va a consistir en la construcción progresiva de un traductor de código C a un código intermedio en otro lenguaje que se pueda procesar mediante un intérprete. Para ello se os proporcionará documentación y código ya realizado que deberéis completar. La práctica final se abordará a través de sucesivas aproximaciones guiadas, hasta llegar a la propuesta definitiva.

En la primera aproximación vamos a aumentar la calculadora desarrollada en las sesiones 1-5, de forma que la evaluación de las expresiones no se haga de forma directa, si no a través de la generación de código en notación postfija (polaca inversa, RPN) correspondiente al lenguaje Forth, que será procesado por un intérprete de dicho lenguaje. Esto nos permitirá transferir parte de los problemas técnicos (uso de variables de nombre arbitrario, almacenado de sus valores, por ejemplo) al lenguaje destino.

En sucesivos pasos iremos convirtiendo la calculadora en un traductor, incluyendo estructuras de control, y otras funcionalidades para disponer de un lenguaje de alto nivel más completo.

Trabajo a realizar:

Para esta primera aproximación se pide únicamente la traducción de expresiones aritméticas y de asignaciones a variables de una notación infija como en el Lenguaje C a notación postfija que serán evaluadas por un intérprete de Forth.

1. Leed el enunciado **completo** antes de comenzar.
2. Instalad *gforth* siguiendo las instrucciones incluidas al final del enunciado y probad los ejemplos explicados en el apartado anterior.
3. Descargad el fichero **`calc5i.y`** y renombradlo a **`calc6.y`**
4. Adaptad **`calc6.y`** para que traduzca las expresiones introducidas (sin tener en cuenta las variables) en código Forth que pueda ser evaluado en el intérprete ***gforth***.
5. Añadid lo necesario para incluir variables dentro de las expresiones. En Forth hay que crear las variables antes de usarlas, pero sólo se puede hacer una vez.
6. Abordad la asignación de un valor a una variable como tarea final.
7. Evaluad resultados usando ***gforth***. Para ello podéis:
   a. editar en un fichero (por ejemplo, codigo.txt) una serie de expresiones
   b. ejecutar o bien **`cat codigo.txt | ./calc6 >res.txt; cat res.txt | ./gforth`**
   c. o bien **`cat codigo.txt | ./calc6 | ./gforth`**
8. Alternativamente se puede usar el intérprete ***gforth*** online: http://nhiro.org/learn_language/FORTH-on-browser.html

Entrega:

Seguid las instrucciones disponibles en el entregador de AG.

Después de hacer la entrega, descargad vuestro fichero y comprobad que funciona correctamente y que cumple con las indicaciones.

Primera Aproximación

Esta primera parte de la práctica consiste en transformar la calculadora de la sesión 5 (código adaptado para operar con enteros, calc5i.y). En su versión original tratamos con un intérprete que evalúa expresiones. Ahora se pide convertirlo en un traductor, es decir, en vez de calcular el valor de la expresión debe transformar la expresión del formato de entrada habitual (infijo) en otro formato que corresponde al lenguaje Forth. En lo que respecta a ésta primera parte de la práctica, la característica a tener en cuenta de dicho lenguaje es que se basa en la notación postfija (polaca inversa).

Por tanto, la traducción de los números y operadores debe ser literal, sólo cambia la notación, en ningún caso se realiza una evaluación.

Vamos a incluir un operador adicional (un punto) que es el que representa la acción de imprimir un valor. Es decir, la acción habitual de imprimir el resultado de evaluar la expresión cuando se alcanza el salto de línea, se sustituirá (traducirá) por la impresión con un punto.

Una expresión como 1+2*3 \n se puede traducir como

```
1 2 3 * + .
```

```
2 3 * 1 + .
```

```
3 2 * 1 + .
```

Cualquiera de estas opciones es válida y cumple con las precedencias esperadas entre los operadores + y *. Se deja al alumno determinar la secuencia correcta desde el punto de vista de la asociatividad esperada.

Pensad que si antes las acciones semánticas debían operar sobre los operandos en función de la operación introducida, ahora deben traducir la secuencia de entrada a otro formato. Desde el punto de vista léxico la salida es idéntica a la entrada, tanto los números como los operadores se representan igual, sólo cambia la secuencia.

Un caso particular se produce cuando se lee el salto de línea (intro) en la entrada. En calc5i lleva asociada la acción semántica de imprimir el resultado de la expresión evaluada. Como ahora ya no se pide una evaluación de la expresión, sino su traducción, deberemos traducir también de alguna forma la acción de imprimir. Eso lo hacemos con el operador . (punto) que en el lenguaje Forth se emplea como comando de impresión.

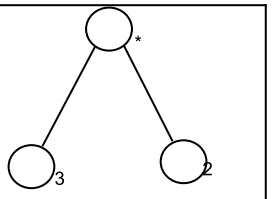
La traducción de la notación infija de entrada a la notación postfija de salida se debe realizar **usando únicamente acciones semánticas que contengan `printf`**. No se deben usar variables adicionales para impresiones temporales, ni ficheros intermedios, etc

Sugerencias: Definir variables necesarias, definir negate para otros intérpretes.

```
main () {
     printf ("variable A  variable B  variable C ... \n") ;
     printf (" : negate 0 swap - ; \n") ;
     printf (" : negate -1 * ; \n") ;
     yyparse ();
}
```

Algunos comentarios y recomendaciones

1. Para que quede claro. Si al introducir en la calculadora **`calc5i`** una expresión como **`3*2<intro>`**, teníamos como resultado el valor 6, lo que se pide ahora **`calc6`** es la salida: **`3 2 * . <intro>`**. La traducción de una expresión más compleja como **`1+2+3<intro>`** debe ser: o bien **`1 2 + 3 + . <intro>`** , o bien **`3 2 1 + + . <intro>`** debido a la asociatividad del operador +. La salida **`1 2 3 + + . <intro>`** es equivalente, pero sólo por el hecho de que el operador + se puede aplicar en un orden cualquiera.
2. La traducción en notación postfija podemos entenderla como un lenguaje especial, es decir una secuencia de instrucciones destinadas a una "máquina de pila". Este tipo de máquina, cpu (o máquina virtual de pila), en vez de operar sobre registros internos, trabaja sobre una pila. Al interpretar la secuencia **`3 2 * .`** lo que hará es tomar el primer elemento (3) de la entrada y, al ser numérico. introducirlo en la pila. El siguiente elemento (2) al ser numérico también lo introduce en la pila situándolo sobre el anterior. A continuación, llega el operador *, y al ser de tipo binario tomará los dos elementos de la cima de la pila (eliminándolos), operará sobre ellos y dejará el resultado sobre la pila. Por último, llega el operador . que es unario. Por lo que toma el elemento cimero de la pila y lo manda al flujo de salida (lo imprime).
3. El proceso de traducción puede que no resulte intuitivo al comienzo, pero es mucho más sencillo de lo que parece, basta con cambiar las acciones semánticas, para que en vez de transmitir valores de operaciones, impriman la expresión en notación postfija.
4. Algunas acciones semánticas serán superfluas, ya que no hay nada que realizar para ciertas producciones. No obstante, **es preferible incluir una acción semántica vacía** `{ ; }` antes que omitirla. Si se omite, se entiende que se va a realizar la operación **`$$ = $1`**, lo cuál no sólo no nos interesa, sino que puede ser causar algún error.
5. Recordamos cuestiones vistas en otras asignaturas (Estructuras de Datos). Recorridos de Árboles Binarios que representan expresiones. Dada la expresión **`3*2`**, la podemos representar con el árbol de la imagen. Fijados en que dicha representación es igual de válida para gramáticas, en este caso para la subgramática de <expresion>.

Para obtener la notación infija de la expresión, empleamos el recorrido: arbol→hijo-izq, arbol→operador, arbol→hijo-dcha. Para obtener la notación postfija, el recorrido es arbol→hijo-izq, arbol→hijo-dcha, arbol→operador. En ambos casos, tenemos que imprimir el contenido de cada nodo cuando accedamos a él.	

6. Si os resulta complicado realizar lo que se pide, podéis empezar por una fase más sencilla: que el traductor imprima a la salida la misma expresión que lee en la entrada (es decir, en notación infija). Una vez resuelto podéis pasar a generar la notación postfija.
7. Volvemos a comentar, en ningún caso se debe modificar código que no sea el correspondiente a las acciones semánticas (el código que va encerrado entre llaves {} en cada producción del fichero calc5i.y)

Breve Introducción a Forth

Aunque el objetivo de la práctica final es realizar un traductor de C a código intermedio, vamos a comenzar por una fase más elemental: traducir las expresiones que evaluamos con nuestra calculadora en código interpretable en Forth.

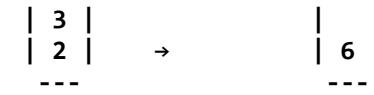
Es decir, es necesario traducir expresiones como

```
2*3<intro>
A=2*3<intro>
B=A*10+23/10<intro>  etc.
```

en un código que sea evaluable por un intérprete de Forth.

Se recomienda recurrir a un intérprete de Forth para ir probando los ejemplos que se plantean aquí. Consultad la última página para instalar **gforth**, o alternativamente al intérprete online: http://nhiro.org/learn_language/FORTH-on-browser.html

La principal particularidad de Forth es que emplea una pila de parámetros sobre la que operan todas las funciones (denominadas *palabras*). Así, la operación de multiplicación * tomará los dos valores de la cima en la pila, para sustituirlos por su producto.



En notación de los así llamados *comentarios de pila*, para la multiplicación tendríamos:

```
(a b – a*b)
```

Para evaluar la expresión **2*3<intro>** en Forth emplearemos la siguiente secuencia:

```
2 3 * . <intro>
```

Los números 2 y 3 se reconocen como tal (*literales*) y son colocados por el intérprete en la pila de parámetros. El operador (*palabra*) * los extrae y los sustituye por el producto de ambos. El operador . es una *palabra* que imprime el valor de la cima de la pila (eliminándolo). Es fácil reconocer que la notación infija a la que estamos habituados (operador binario entre operandos) se ha transformado en notación postfija (polaca inversa: primero los operandos, luego el operador). Quién haya manejado calculadoras de HP estará habituado a esta notación. Adviértase que entre cada uno de los símbolos de entrada debe haber al menos un espacio en blanco. La línea se evalúa al pulsar ***<intro>***, y el intérprete responderá con:

```
6 ok
```

Otros operadores aritméticos son: **+, - y /**. Todos ellos operan de igual forma que la multiplicación. Es importante precisar que los números en la pila son siempre enteros, generalmente con el rango de valores -32768...+32767. La división será por tanto entera y tiene su complemento en el operador `mod` que calcula el resto de una división. Para negar un número existe la *palabra* **negate (a -- -a)**.

Algunas *palabras* habituales de gestión de pila son (entre otras):

- **dup (a -- a a)** duplica el valor de la cima de la pila
- **swap (a b -- b a)** intercambia los dos valores de la cima
- **drop (a b -- a)** elimina el valor de la cima

Cualquiera de las operaciones que se realice sobre una pila con menos operandos de los necesarios provocará un error y, en algunos intérpretes, el vaciado de la pila.

Para definir una variable usaremos la *palabra* **variable**, clasificada en Forth como *palabra constructora*, porque añade una definición al *diccionario* interno. Para crear las variables A y B:

```
variable A  variable B
```

Para realizar la operación **A=6**, usaremos el operador ! .

```
6 A !
```

El intérprete identifica la letra A como una *palabra* definida en el *diccionario* como variable. Así que deja sobre la pila la dirección de memoria asignada a la variable. El operador ! toma de la pila una dirección y un valor, y almacena dicho valor en la dirección. De forma similar podremos resolver **A=3*2** :

```
3 2 * A !
```

Para recuperar el valor de una variable, usaremos el operador **@** :

```
A @ .                    imprimirá:
6 ok
```

La expresión **B=A*10+(23/10)** traducida a Forth debe quedar:.

```
23 10 / A @ 10 * +  B !
B @ .                    imprimirá:
62 ok
```

Aunque no es necesario ahora mismo, podemos dar un pequeño atisbo de la filosofía de este lenguaje. Para construir nuevas *palabras* de tipo función, más adelante usaremos otro *constructor*, la *palabra* : .

```
: cuadrado dup * ;
```

Aquí definimos la *palabra* **cuadrado** que contiene la secuencia **dup *** y el terminador de definición ; . Una aplicación de ésta palabra definida la tenemos en el ejemplo:

```
5 cuadrado .          que es evaluada con
25 ok
```

En C tendríamos la definición equivalente:

```
int function cuadrado (int valor)
{
      return (valor*valor) ;
}
```

Con esto nos podemos ir haciendo una idea de en qué consistiría la traducción de programas en C a Forth. Empezaremos por las expresiones de una calculadora y cambiaremos progresivamente de objetivos.

Recursos para realizar la práctica:

Para aprender C:

- http://atc2.aut.uah.es/~rduran/Informatica/docs/leng_c.pdf

Bison y Flex:

- http://www.gnu.org/software/bison/manual/
- http://flex.sourceforge.net/manual/

Forth

El nivel de conocimiento de Forth necesario para la práctica es elemental, pero dejamos aquí los siguientes enlaces para que sirvan como referencia:

- http://en.wikipedia.org/wiki/Forth_%28programming_language%29 introducción (en inglés)
- http://www.disc.ua.es/~gil/forth.pdf introducción en castellano que más allá de dar una visión del funcionamiento básico, se extiende en los principios de la programación.
- http://www.complang.tuwien.ac.at/forth/gforth/ gforth de GNU.
- http://www.complang.tuwien.ac.at/forth/gforth/Docs-html/ su documentación (en inglés y html)
- http://www.complang.tuwien.ac.at/forth/faq/faq-general.html conjunto de FAQs sobre el lenguaje en sí (en inglés)
- Intérprete gforth online: http://nhiro.org/learn_language/FORTH-on-browser.html

Instrucciones para instalar y comenzar con gforth:

En http://www.complang.tuwien.ac.at/forth/gforth/ están disponibles los fuentes y binarios necesarios para instalar un intérprete de Forth. La instalación en Linux debería ser sencilla siguiendo estos pasos:

- Descargar el fichero **`gforth-0.7.3.tar.gz`** (la versión anterior también debería funcionar, pero da problemas en las aulas informáticas).
- Descomprimir con
  - **`gzip -d gforth-0.7.3.tar.gz`**
  - **`tar xvf gforth-0.7.3.tar`**

  o en caso de error probar con
  - **`tar xvf gforth-0.7.3.tar.gz`**
- en el directorio **`gforth-0.7.3`** se ejecuta una configuración con **`./configure`**
- se compila el proyecto con **`make`** y se crean varios intérpretes/compiladores.
- Se puede usar **`gforth`** con el comando **`./gforth`**. Para salir del intérprete, hay que emplear la *palabra* **`bye`**.
- Para trasladar **`gforth`** de directorio hay que copiar también la imagen **`gforth.fi`**.

En Ubuntu podéis usar **`sudo apt-get install gforth`**