Universidad de La Laguna

ESIT – INFORMÁTICA
Departamento de Ingeniería Informática y de Sistemas

Computabilidad y Algoritmia
Curso 2019-2020

PRACTICA 5: Expresiones Regulares

5.1. Objetivos

- Consolidar los conocimientos adquiridos sobre Expresiones Regulares, sus operadores y significado.
- Conocer las diferentes formas de una expresión: infija, postfija, prefija.
- Implementar en C++ estructuras de datos tales como pilas o árboles.
- Practicar en C++ el uso de clases genéricas a través de las plantillas (*templates*).
- Profundizar en las capacidades de diseñar y desarrollar programas orientados a objetos en C++.

5.2. Introducción

Las expresiones regulares (ER) constituyen un mecanismo de especificación de lenguajes regulares: toda expresión regular representa un lenguaje, de modo que sirve para representar todas las cadenas pertenecientes a ese lenguaje. Las ERs se utilizan de forma habitual en algoritmos de búsquedas en cadenas textuales (“buscar” o “buscar y reemplazar”) en editores y procesadores de textos. El concepto de expresión regular surgió en los años 1950 cuando el matemático Stephen Cole Kleene formalizó la descripción de los lenguajes regulares. Las ERs pasaron a usarse de modo habitual con las utilidades de procesamiento de texto de Unix (`grep`, `sed` o `awk` son algunas de ellas). Desde la década de 1980 se han propuesto diferentes sintaxis para la escritura de ERs. Entre las más destacables se encuentran el estándar POSIX y la sintaxis de Perl [1].

Las expresiones regulares se utilizan en los motores de búsqueda, en los procesadores y editores de texto, en las utilidades de procesamiento de texto como `sed` o `AWK` y en la etapa de análisis léxico de los compiladores. Muchos lenguajes de programación proporcionan capacidades *regex* ya sea incorporadas o a través de librerías.

La forma habitual de escribir expresiones, tanto aritméticas como expresiones regulares es la que se conoce como notación infija [2]. Una forma alternativa es la

notación postfija [3] también conocida como notación polaca inversa. Así por ejemplo, la expresión postfija equivalente a la expresión regular infija

```
(a|b)*xz*
```

sería la expresión:

```
ab|*x&z*&
```

Nótese que se introduce explícitamente el operador `&` para indicar la concatenación, de modo que los operadores de expresiones regulares a usar en la forma postfija serían `&`, `|` , `*` para concatenación, disyunción y repeticiones respectivamente.

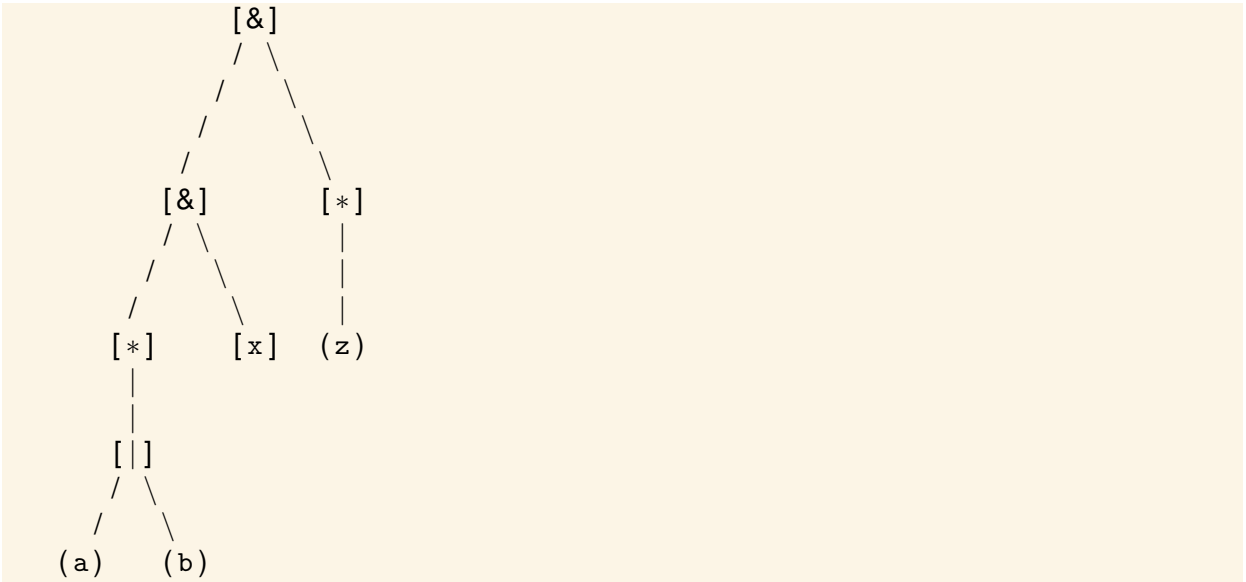
El Algoritmo 1 permite convertir una expresión de infija a postfija utilizando una pila y una traza del funcionamiento del algoritmo se puede ver en [4] para el caso de una expresión aritmética.

1. Los operandos se imprimen conforme se leen en la entrada;
2. Si la pila está vacía o contiene un paréntesis abierto en el tope, apilar el operador entrante;
3. Si el símbolo entrante es un paréntesis abierto, apilarlo;
4. Si el símbolo entrante es un paréntesis cerrado, desapilar e imprimir los operadores hasta hallar un paréntesis abierto. Descartar el par de paréntesis;
5. Si el símbolo entrante tiene mayor precedencia que el que se encuentra en el tope de la pila, apilarlo;
6. Si el símbolo que se lee tiene la misma precedencia que el que se encuentra en el tope de la pila, usar asociación. Si la asociaciación es de izquiera a derecha, desapilar e imprimir el elemento desapilado y apilar el operador que acaba de leerse. Si la asociatividad es de derecha a izquierda, apilar el operador que acaba de leerse.;
7. Si el símbolo entrante tiene menor precedencia que el del tope de la pila, desapilar e imprimir el operador desapilado. Comprobar el operador entrante con el nuevo top de la pila;
8. Al final de la expresión, sacar de la pila todos los operadores e imprimirlos (no deberían quedar paréntesis);

Algoritmo 1: Conversión de infija a postfija

Una vez convertida en postfija, resulta sencillo construir el árbol de la expresión regular. El árbol de una expresión [5] es una forma de representación de la misma que resulta útil en diversos contextos. El Listado 5.1 muestra el árbol correspondiente a la expresión regular `(a|b)*xz*`.

Para implementar un árbol binario en C++ basta usar una estructura de datos (Clase) que contenga tres elementos: el dato que se almacena en cada nodo del árbol (que puede en sí mismo ser una estructura compleja) y sendos punteros (hijo izquierdo, hijo derecho) que apuntarán a los subárboles del nodo en cuestión. A veces se incluye un tercer puntero al nodo padre del considerado. La referencia *C++ Generic Binary Search Tree* [6] puede servir como guía de esta implementación. Téngase en cuenta que los nodos del árbol de una expresión regular pueden ser binarios o unarios, dependiendo de la aridad del operador del nodo.



Listado 5.1: Árbol correspondiente a la expresión regular `(a|b)*xz*`

El Algoritmo 2 permite construir el árbol de la expresión postfija usando nuevamente una pila, pero en este caso una pila de árboles (nodos).

1. Crear una pila;
2. Leer el siguiente símbolo de la expresión postfija;
3. Si el símbolo es un operando, cree un nuevo nodo del árbol que represente a ese operando y apile (push) ese nodo;
4. Si el símbolo es un operador, saque (pop) de la pila dos árboles (A1 y A2). Cree un nuevo árbol con el operador como raíz y A1 y A2 como hijos. Apile este nuevo árbol en la pila;
5. Repita este proceso hasta que se complete la lectura de la expresión postfija;
6. Al final del proceso la pila contendrá un único árbol correspondiente a la expresión;

Algoritmo 2: Obtención del árbol de una expresión postfija

Una vez que se tiene el árbol de la expresión, es posible obtener las distintas formas de la misma (prefija, infija, postfija) recorriendo de forma correspondiente el árbol [5].

5.3. Ejercicio práctico

Desarrollar un programa `ER2Tree.cpp` que lea un fichero de texto en el que figura una relación de expresiones regulares escritas en notación infija y que usan el alfabeto $\Sigma = \{a, b, c, \ldots, z, *, |, (,)\}$, es decir letras en minúsculas y los operadores estrella y disyunción (en las ERs de entrada, la concatenación es un operador implícito). El fichero de entrada contendrá una única ER en cada línea. Al ejecutarse `ER2Tree` de la forma:

```
$ ./ER2Tree filein.txt fileout.txt
```

se creará el fichero de salida `fileout.txt` en el cual figurará una línea por cada ER del fichero de entrada. En cada línea del fichero de salida se deberá imprimir, separadas por tabuladores, las 3 formas de notación de la ER en el orden: infija, postfija, prefija y cada línea finalizará con un número natural que corresponde con el número de nodos del árbol de la expresión, tal como se muestra en la Tabla 5.1. Al escribir la expresión en infija (primera columna de la tabla) no se explicita el operador de concatenación, del mismo modo que no se hace en las ERs del fichero de entrada.

Infija	Postfija	Prefija	Nodos
a|b	ab|	|ab	3
a|bc	abc&|	|ab&c	5
(a|b)(c|d)	ab|cd|&	&|ab|cd	7

Tabla 5.1: Ejemplo de salida del programa

5.4. Rúbrica de evaluación del ejercicio

Se señalan a continuación los aspectos más relevantes (la lista no es exhaustiva) que el profesorado tendrá en cuenta a la hora de evaluar el trabajo que el alumnado presentará en la sesión de evaluación de la práctica:

- El comportamiento del programa debe ajustarse a lo solicitado en el enunciado.
- Capacidad del programador(a) de introducir cambios en el programa desarrollado.
- El programa diseñado ha de seguir el paradigma OOP. Identifique clases y objetos que permitan modelar adecuadamente el escenario de trabajo que se propone.
- Modularidad: el programa ha de escribirse de modo que las diferentes funcionalidades que se precisen sean encapsuladas en métodos cuya extensión textual se mantenga acotada.
- El programa ha de ceñirse forzosamente al formato de escritura de programas adoptado en las prácticas de la asignatura.

Si el alumnado tiene dudas respecto a cualquiera de estos aspectos, debiera acudir al foro de discusiones de la asignatura para plantearlas allí. Se espera que, a través de ese foro, el alumnado intercambie experiencias y conocimientos, ayudándose mutuamente a resolver dichas dudas. También el profesorado de la asignatura intervendrá en las discusiones que pudieran suscitarse, si fuera necesario.

Bibliografía

[1] Análisis Léxico y Sintáctico. Expresiones Regulares en Flex, Lex y Perl. https://campusvirtual.ull.es/ocw/pluginfile.php/2209/mod_resource/content/0/perlexamples/perlexamples.pdf

[2] Infix notation https://en.wikipedia.org/wiki/Infix_notation

[3] Reverse Polish notation https://en.wikipedia.org/wiki/Reverse_Polish_notation

[4] Traza del algortimo http://csis.pace.edu/~wolf/CS122/infix-postfix.htm

[5] Expression tree https://en.wikipedia.org/wiki/Binary_expression_tree

[6] C++ Generic Binary Search Tree https://gist.github.com/lawliet89/8623547