## Examen práctica PRO2 Q1-2019-2020–Problema 2 X67036_es

1. El peso de este ejercicio en la nota del exámen de la práctica es de un 40%
2. Este problema tiene exclusivamente nota automática, es decir, el peso de la nota manual es 0% y el de la nota automática 100% (10 puntos/10).
3. El peso de los juegos de pruebas público y privados en el cálculo de la nota automática es idéntico para todos ellos (cada juego de pruebas superado 2.5/10 puntos de la nota automática de este problema).

Se nos da un árbol binario cuyos nodos internos (los que tienen dos subárboles no vacíos) contienen o bien un operador de unión o bien un operador de intersección, y cuyas hojas (nodos con ambos subárboles vacíos) contienen cada una un subconjunto finito de strings. De este modo un árbol representa una expresión conjuntista: los operandos son subconjuntos finitos y los operadores son uniones e intersecciones.
Para representar el árbol se utiliza la siguiente definición:

```
typedef set<string> subcjt;
struct nodo {
   string tipo; //  "H" == nodo hoja, "I" == intersección, "U" == unión
   subcjt c; // si tipo == "H"  entonces c guarda un subconjunto
                 // finito {c0,c1, ... } de strings
                 // en orden ascendente c0 < c1 < c2 < ...
};
typedef BinTree<nodo> expr_cjt;

// Pre: todo nodo en el árbol a tiene exactamente 0 o exactamente
//      2 subárboles vacíos; los nodos de grado 2 son de tipo == "U"
//      o de tipo == "I", los nodos
//      de grado 0 (hojas) son de tipo "H" y contiene cada una
//      un subconjunto finito de strings según la
//      descripción dada arriba

// Post: devuelve el subconjunto resultante de evaluar la expresión conjuntista
//       representada por el árbol

subcjt evalua_expresion_cjt(const expr_cjt& a);
```

Se os suministra un módulo (ficheros `subset-tree-IO.o` y `subset-tree-IO.hh`) con las operaciones de entrada/salida para los árboles y para subconjuntos de strings. También os damos un módulo funcional `ops-cjts` (ficheros `.hh` y `.o`) que os dan las operaciones de unión e intersección de conjuntos ya implementadas y compiladas. En particular, el módulo `subset-tree-IO` os proporciona los operadores `>>` para leer un conjunto de strings o una expresión conjuntista (que están escritos siguiendo el formato del apartado de **Entrada**), y un operador `<<` para imprimir un conjunto de strings (según el formato de **Salida**).

Con todo ello completaréis un pequeño programa que lee árboles, evalúa las expresiones que éstos representan e imprime los conjuntos resultantes. Utilizad el fichero `plantilla_solucion.cc.txt` para escribir vuestra solución. Vuestra solución debe hallarse en el fichero `solution.cc`, que es el único fichero a enviar (una buena idea es renombrar el fichero `plantilla_solucion.cc.txt` como `solution.cc`).

## Entrada

La entrada comienza en un número $n \geq 0$. A continuación viene una secuencia de $n$ expresiones conjuntistas en notación infija. Cada expresión está delimitada por las string `BEGIN_EXPR` ...`END_EXPR`. Los conjuntos (operandos) se introducen delimitando sus contenidos mediante llaves de apertura y cierre '{' '}'. Debe haber al menos un blanco entre la llave de apertura y el primer elemento, y de igual modo debe existir al menos un blanco entre el último elemento y la llave de cierre. Se usan las strings `U` e `I` para denotar uniones e intersecciones, respectivamente. Las intersecciones son más prioritarias que las uniones y puede usarse paréntesis para modificar el orden de evaluación de las expresiones.

## Salida

Se escribe el conjunto de strings resultante de evaluar cada árbol dado en la entrada, con blancos separando los elementos del conjunto y finalizando con un salto de línea.

## Ejemplo de entrada 1

```
5
BEGIN_EXPR { } END_EXPR
BEGIN_EXPR { tres cuatro uno dos nueve } END_EXPR
BEGIN_EXPR { gat gos } U { cavall conill gat } END_EXPR
BEGIN_EXPR { gat gos } I { cavall conill gat } END_EXPR
BEGIN_EXPR ( { uno ocho tres doce } U { nueve uno tres } ) I { cuatro ocho nueve } END_EXPR
```

## Ejemplo de salida 1

```
{ }
{ cuatro dos nueve tres uno }
{ cavall conill gat gos }
{ gat }
{ nueve ocho }
```

## Información del problema

Autoría: Profesores de PRO2

Generación: 2026-01-25T17:38:23.571Z