Universitat Politècnica de Catalunya
Facultat d'Informàtica de Barcelona

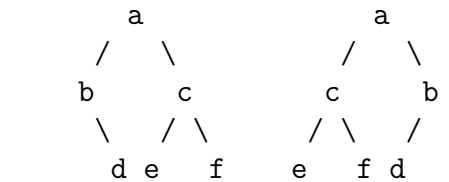
Cognoms, Nom | D.N.I.

Titulació: Grau en Enginyeria Informàtica
Assignatura: Programació 2 (PRO2)
Duració: 1h 30m

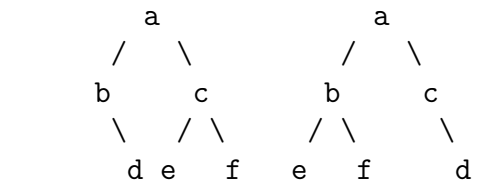
Curs: Q2 2019–2020 (1r Parcial)
Data: 27 d'abril de 2020

1. **(2 puntos) BinTrees isomorfos**
   Tiempo estimado: 10 minutos

   Diremos que dos BinTree son isomorfos si hay alguna manera de intercambiar arboles izquierdos y derechos de uno de ellos tantas veces como sea necesario de tal manera que los dos árboles sean iguales. Por ejemplo



   son isomorfos, pero



   no lo son.

   Completa la función dada más abajo para que cumpla su especificación, es decir, indica qué instrucciones deben reemplazar los lugares indicados con números (`[---n---]`).

```
bool isomorfos(const BinTree<int>& a, const BinTree<int>& b) {
    bool res;
    if (a.empty() or b.empty()) { [--------- 1 -------------] }
    else if (a.value() != b.value()) { [ ------- 2 ----------] }
    else { [------- 3 ------- ] };
    return res;
}
```

Cada uno de los lugares señalados (1, 2 y 3) consiste en una o más instrucciones consecutivas y simples del estilo `res = ....;`

SOLUCIÓ:

```
1: res = a.empty() and b.empty();
2: res = false;
3: res = isomorfos(a.left(), b.left()) and isomorfos(a.right(), b.right());
   res = res or isomorfos(a.left(), b.right()) and isomorfos(a.right(), b.left());
```

Cognoms, Nom D.N.I.

2. **(1.5 puntos) Invertir prefijo de lista**
Tiempo estimado: 15 min

Completa la función dada más abajo para que cumpla su especificación, es decir, indica qué instrucciones o expresiones deben reemplazar los lugares indicados con números (`[---n---]`).

```
// Pre: it=IT apunta a un elemento x_i de l o it == l.end()
// y l = L = [x_1,....,x_N] donde N >= 0
void reverse_prefix(list<int>& l, list<int>::iterator it) {
  list<int>::iterator p = [------- 1 --------];
  while ([------ 2 ------]) {
    [---- 3 -----];
  };
};
// Post: l está formada por el prefijo invertido de L
// entre L.begin() e IT, y a continuación, L[IT:);
// esto es, l = [x_{i-1},...,x_1,x_i,...,x_N]
```

En una solución correcta

1: es una expresión simple

2: es una expresion booleana

3: es una o más instrucciones simples

y el coste del bucle es proporcional al número de elementos que tiene el prefijo de L que se ha de invertir. Además está prohibido añadir expresiones o instrucciones fuera de los lugares indicados (p.e. no se pueden añadir más inicializaciones o instrucciones al terminar el bucle) y se valorará negativamente cualquier solución que modifique `*it` o `*p` (por ejemplo, intercambiar `*p` con `*it`).

SOLUCIÓ:
Algunas soluciones correctas:

```
1: l.begin()
2: p != it
3: l.insert(l.begin(), *p); // o l.push_front(*p);
   p = l.erase(p);
```

Alternativa #1

```
1: y 2: como arriba
3: it = l.insert(it, *p); // o l.insert(it, *p); --it;
   p = l.erase(p);
```

Alternativa #2

```
1: it
2: p != l.begin()
3: p = l.insert(p, *(l.begin())); // o l.insert(p, *(l.begin())); --p;
   l.erase(l.begin()); // o l.pop_front();
```

Cognoms, Nom D.N.I.

3. **(2 puntos) Partición de listas**
Tiempo estimado: 15 minutos
Tenemos que diseñar un procedimiento `parte_lista` que, dados una lista `l` de enteros

y un valor entero `x`, modifica la lista `l` para que todos los elementos menores o iguales que `x` estén delante de todos los elementos mayores que `x` y nos devuelve un iterador al primer elemento $>$ `x`, o a `l.end()` si no hay ningun elemento $>$ `x` en `l`. El orden relativo entre los elementos de `l` no importa y no tiene porque coincidir con el que tuvieran en L.

```
// Pre: l = L
list<int>::iterator parte_lista(list<int>& l, int x);
// Post: l es una permutación de L, it = parte_lista(l,x)==l.end()
// si no hay elementos mayores que x en L o it apunta a un elemento > x,
// todos los elementos en l[it:) son > x, y todos los elementos
// de l[:it) son <= x
```

Tu solución ha de ser iterativa y debe preservar necesariamente este invariante:

- `l` es una permutación de L **y**
- todos los elementos de `l[:it1)` son menores o iguales que `x` **y**
- todos los elementos de `l[it2:)` son mayores que `x`

siendo L el valor original de la lista `l` e `it1` e `it2` iteradores a elementos de `l`.

SOLUCIÓ:

```
// Pre: l = L
list<int>::iterator parte_lista(list<int>& l, int x) {
  list<int>::iterator it1 = l.begin();
  list<int>::iterator it2 = l.end();
  while (it1 != it2) {
    if (*it1 <= x) ++it1;
    else {
      it2 = l.insert(it2, *it1); // o l.insert(it2, *it1); --it2;
      it1 =  l.erase(it1);
    }
  }
  return it2;
}
// Post: l es una permutación de L, it = parte_lista(l,x)==l.end()
// si no hay elementos mayores que x en L o it apunta a un elemento > x,
// todos los elementos en l[it:) son > x, y todos los elementos
// de l[:it) son <= x
```

Cognoms, Nom D.N.I.

4. **(2 puntos) Substitució en un vector**
   Temps estimat: 15 minuts
   Escriu una funció `subst_sum` que, donat un vector `v` d'enters $\texttt{v} = (v_0, \ldots, v_(n-1))$ no

   buit, substitueix el primer element del vector per la suma de tots els seus elements. La teva solució ha de utilitzar una funció d'immersió `i_subst_sum` recursiva. Implementa la funció d'immersió, i implementa la funció `subst_sum` utilitzant la funció `i_subst_sum`, respectant les especificacions donades:

```
// Pre: v = (V_0,...,V_{N-1}) i N > 0
void subst_sum(vector<int>& v);
// Post: v = (S,V_1,...,V_{N-1}), ón S = V_0 + ··· + V_{N-1}

// Pre: v = (V_0,...,V_{N-1}), N > 0, 0 ≤ i < N,
// s = V_0 + ··· + V_{i-1}
void i_subst_sum(vector<int>& v, int i, int& s);
// Post: v = (s,...,V_{N-1}),
// s = V_0 + ··· + V_i + ··· + V_{N-1}
```

SOLUCIÓ:

```
// Pre: v = (V_0,...,V_{N-1}) i N > 0
void subst_sum(vector<int>& v) {
  int s = 0;
  i_subst_sum(v, 0, s);
}
// Post: v = (S,V_1,...,V_{N-1}), ón S = V_0 + ··· + V_{N-1}

// Pre: v = (V_0,...,V_{N-1}), N > 0, 0 ≤ i ≤ N,
// s = V_0 + ··· + V_{i-1}
void i_subst_sum(vector<int>& v, int i, int& s) {
  if (i == v.size() - 1) v[0] = s + v[i];
  else {
    s += v[i];
    i_subst_sum(v, i+1, s);
  }
}
// Post: v = (s,...,V_{N-1}),
// s = V_0 + ··· + V_i + ··· + V_{N-1}
```