Primer examen parcial de PRO2 Durada: 2h 30m 16/11/2017

Cognoms **Nom** **DNI**

Observació: Es recomana llegir tots els apartats abans de començar a resoldre cada problema.

Problema 1 (5 punts)

1.1 Feu un disseny recursiu de la funció `pos_insertar_rec`. Donats un enter `x` i una llista d'enters `sel` ordenada en ordre creixent, `pos_insertar_rec` retorna un iterador que apunta a l'element de `sel` davant el qual s'ha d'inserir `x` perquè `sel` segueixi estant ordenada en ordre creixent després de la inserció, si aquest element existeix en `sel`, o un iterador que apunta a sel.end(), en un altre cas. Per exemple, si `x` és `6` i `sel` és `{1, 2, 4, 5, 7, 8}`, el resultat de la crida `pos_insertar_rec(x,sel)` ha d'apuntar a `7`.

En primer lloc es demana completar l'especificació i la implementació de la funció d'immersió que fareu servir per realitzar un disseny recursiu de la funció `pos_insertar_rec`. Concretament, heu d'especificar els paràmetres i/o resultats addicionals, la precondició, la postcondició i el cos de la funció d'immersió `i_pos_insertar_rec`. A l'apartat **1.2** podeu veure l'especificació de `pos_insertar_rec`. **[2.5 punts]**

i_pos_insertar_rec(**int** *x*, *list*<**int**>& *sel*, ...); // paràmetres addicionals

/* Pre:

*/

/* Post:

*/

// cos de la funció

1.2 Completeu la implementació de la funció `pos_insertar_rec` utilitzant la funció d'immersió que heu dissenyat a l'apartat anterior. **[0.5 punts]**

```
list <int>:: iterator  pos_insertar_rec (int x, list <int>& sel) {
/* Pre: sel està ordenada en ordre creixent. */
/* Post: El resultat apunta a l'element de sel davant del qual s'ha d'inserir x
perquè sel segueixi estant ordenada en ordre creixent després de la inserció, si existeix
aquest element; en un altre cas, el resultat apunta a sel.end(). */
  [                                                                    ]
}
```

1.3 Utilitzant la funció `pos_insertar_rec` de l'apartat **1.2**, completeu la implementació i la invariant de l'acció següent. **[2 punts]**.

```
void seleccionados (int n, list <int>& sel) {
/* Pre: Al canal estàndard d'entrada hi ha una seqüència de nombres naturals no repetits
   acabada en 0; n > 0; sel = SEL; i SEL és buida. */
/* Post: sel conté els n nombres més grans de la seqüència ordenats en ordre creixent. */
   int x;
   cin >> x;
   /* Inv: La part tractada de la seqüència té una longitud m.
   [                                                                    ]
   */
   while (x ≠ 0) {
     bool insertar ;
     list <int>:: iterator it ;
     if (sel.empty()) {
       [                                                                ]
     }
     else if ( [                                                ] ) {
       [                                                                ]
     }
     else {
       [                                                                ]
       [                                                                ]
       [                                                                ]
     }
     if ( insertar ) sel . insert ( it ,x );
     cin >> x;
   }
}
```

Cognoms	Nom	DNI

Problema 2 (5 punts)

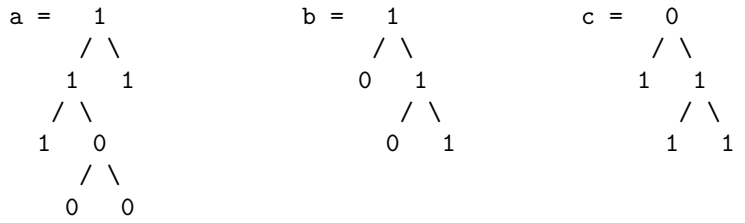
Caràcter compatible en un arbre filogenètic.

Donat un arbre de nodes amb valors 0 i 1 únicament, volem saber si aquests valors són compatibles. Diem que els valors d'un arbre binari A són *compatibles* si compleixen les dues condicions següents:

1. Si A conté algun node amb valor 1, llavors tots els nodes amb valor 1 de A formen un únic arbre.
2. Si A conté algun node amb valor 0, llavors tots els nodes amb valor 0 de A formen un únic arbre.

Treballarem amb la restricció que cada node de l'arbre A o bé té 0 fills no buits o bé té 2 fills no buits.

Per exemple, els valors de l'arbre a són compatibles. En canvi els valors dels arbres b i c no ho són. Els nodes amb valor 0 de l'arbre b formen dos arbres en lloc d'un únic arbre. De la mateixa manera, els nodes amb valor 1 de l'arbre c formen dos arbres.



A continuació donem l'especificació del problema que volem resoldre.

```
bool compatible(const BinTree<int>& a);
// Pre: Els nodes de a són 0's o 1's. Els nodes de a tenen o bé
// 0 fills no buits o bé 2 fills no buits.
// Post: El resultat indica si els nodes de a són compatibles.
```

Problema 2.1 (1 punt): Suposem que l'arbre A té el valor x a l'arrel i que A_1 i A_2 són els seus fills esquerre i dret. Expliqueu la compatibilitat de A en funció de la compatibilitat de A_1 i A_2, i en funció d'altres quantitats calculades per a A_1 i A_2. Podeu utilitzar els casos següents:

1. El valor x és diferent dels valors de les arrels de A_1 i A_2. Això és el que passa a l'exemple c.
2. El valor x és igual al valor de l'arrel d'un dels fills de A. Això és el que passa al fill esquerre de l'exemple a i a l'exemple b.
3. El valor x és igual a les arrels de A_1 i A_2. Això és el que passa a l'arrel de l'exemple a.

Problema 2.2 (0.5 punts): Tenint en compte l'apartat **2.1**, completeu la capçalera i l'especificació següents.

```
bool i_compatible(const BinTree<int>& a, un o més paràmetres);
// Pre: Els nodes de a són 0's o 1's. Els nodes de a o bé tenen
// 0 fills no buits o bé tenen 2 fills no buits (i, si cal,
// propietats dels paràmetres nous ...).
// Post: El resultat indica si els nodes de a són compatibles,
// (i, si cal, propietats dels paràmetres nous ...).
```

Problema 2.3 (0.5 punts): Heu de programar l'operació *compatible* fent servir la funció d'immersió *i_compatible*.

Problema 2.4 (2.5 punts): Implementeu la funció *i_compatible*. És important que aquesta implementació sigui lineal en el nombre de nodes de *a*. Si la vostra implementació requereix alguna funció auxiliar, aquesta funció també l'heu d'especificar i implementar.

Problema 2.5 (0.5 punts): Escriviu les hipòtesis d'inducció que necessita el codi de la vostra implementació de *i_compatible*.