Universitat Politècnica de Catalunya
Facultat d'Informàtica de Barcelona

Cognoms, Nom **(Amb lletres majúscules)** D.N.I.

Titulació: Grau en Enginyeria Informàtica **Curs:** Q2 2021–2022 (2n Parcial)
Assignatura: Programació 2 (PRO2) **Data:** 9 de juny de 2022
Duració: 2h 30m

1. **(5 punts)** Diem que en una llista un element és frontissa si és igual que la diferència entre la suma dels elements que el segueixen i la suma dels elements que el precedeixen. Per exemple, a la llista $[1, 2, \underline{1}, \underline{0}, 4]$ els elements subratllats són elements frontissa.

   Ens donen la següent representació amb apuntadors i memòria dinàmica per a una classe llista d'enters amb punt d'interès (`act`), encadenament doble i sense sentinella.

```
class Llista {
private:
    struct node_llista{
        int info;
        node_llista* seg;
        node_llista* ant;
    };
    int longitud;
    node_llista* primer_node;
    node_llista* ultim_node;
    node_llista* act;
    ... // operacions privades
public:
    ... // operacions públiques
}
```

   Es demana:

   (a) (2,5 punts) Implementeu un mètode privat `elimina_node` que donat un punter no nul p a un element de la llista implícita, elimina l'element apuntat per p—i actualitza els atributs de la llista implícita com convingui per a restaurar l'invariant de representació, e.g., `primer_node` apunta al primer element de la llista implícita si no és buida o és `nullptr` si la llista fos buida, etc.

```
// Pre: p ≠ nullptr apunta a un element de la llista implícita i act == nullptr
// Post: La llista implícita conté el resultat d'eliminar l'element
// apuntat per p, i els seus atributs s'han actualitzat convenientment
void elimina_node(node_llista* p);
```

   (b) (2,5 punts) Implementeu un nou mètode `elimina_frontisses`, que elimini els elements frontissa de la llista implícita, amb la següent especificació:

```
// Pre: El paràmetre implícit conté la llista no buida L
// Post: La llista implícita conté el resultat d'eliminar d'L
// els seus elements frontissa i act == nullptr
void elimina_frontisses();
```

SOLUCIÓ:

(a)

```
// Pre: p ≠ nullptr apunta a un element de la llista implícita i act == nullptr
// Post: La llista implícita conté el resultat d'eliminar l'element
// apuntat per p, i els seus atributs s'han actualitzat convenientment
void elimina_node(node_llista* p) {
  if (longitud == 1) { ________________ ;
  } else if ( ________________ ) { // s'ha d'eliminar l'últim
    ________________ ; ________________ ;
  } else if ( ________________ ) { // s'ha d'eliminar el primer
    ________________ ; ________________ ;
  } else { // cas general
    ________________ ; ________________ ;
  }
  ________________ ; ________________ ;
}
```

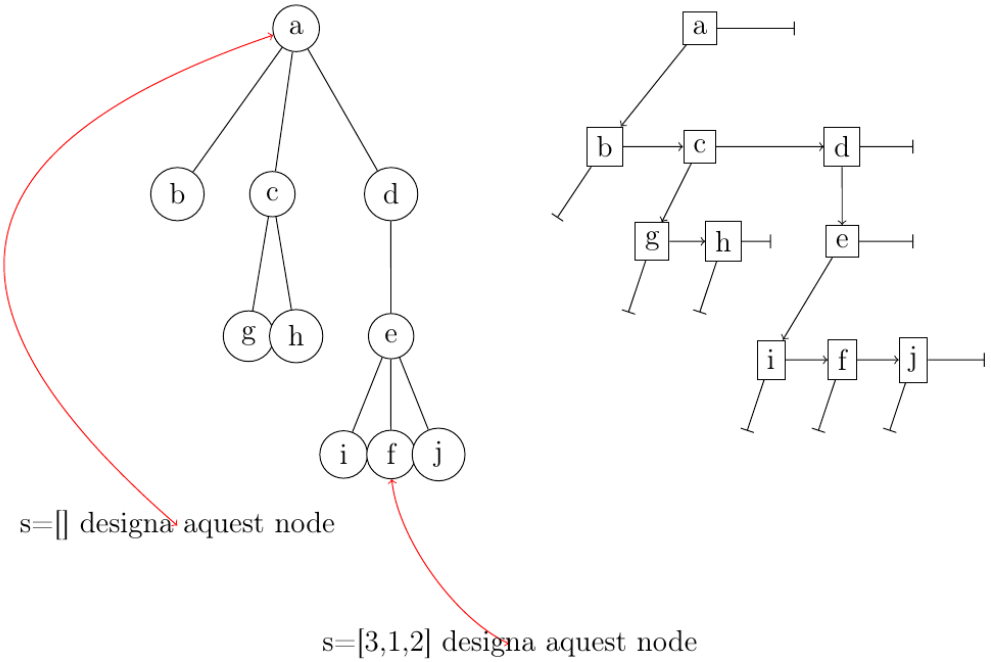
(b)

```
// Pre: El paràmetre implícit conté la llista no buida L
// Post: La llista implícita conté el resultat d'eliminar d'L
// els seus elements frontissa i act == nullptr
void elimina_frontisses() {
    act = ________________ ;
    node_llista* aux = ________________ ;
    int sumaant = ________ ; int sumapost = ________ ;
    for (node_llista* n = ________ ; ________ ; ________ ) {
      ________________ ;
    }
    while ( ________________ ) {
      node_llista* auxseg = aux -> seg;
      int auxinfo = aux -> info;
      if (auxinfo == ________________ ) {
        ________________ ;
      }
      ________________ ; ________________ ;
      ________________ ;
    }
}
```

2. **(5 punts)** Tenim implementada la classe `ArbreGen` mitjançant la representació primer fill-següent germà. Recordem que si un arbre consisteix en un únic node (és una fulla) llavors la seva alçària és 1, altrament l'alçària és 1 + la màxima alcària d'un dels fills de l'arrel. L'alçària d'un arbre buit és, per conveni, zero.

```
template <typename T>
class ArbreGen {
private:
   struct node_arbreGen {
     T info;
     node_arbreGen* primer_fill;
     node_arbreGen* seg_germa;
   };
   node_arbreGen* arrel;
   // capçaleres de mètodes privats
   ...
public:
   ...
};
```

Donat un node x d'un arbre general els seus fills es numeren d'1 en endavant (si x és una fulla no té fills). Suposem que a és un arbre general d'alçària h i talla $n > 0$. Una seqüència d'enters $s = [s_1, s_2, \ldots, s_\ell]$, $\ell < h$, pot designar o no un node específic d'un arbre a, si compleix certes condicions addicionals; en tal cas, es diu que la seqüència és *vàlida*. Suposem que $s' = [s_1, \ldots, s_i]$, $i < h-1$, és una seqüència vàlida que designa a un node x' de l'arbre a. Si s_{i+1} és un número entre 1 i el grau d'x' llavors la seqüència $s = [s_1, \ldots, s_i, s_{i+1}]$ és una seqüència vàlida i designa a l'arrel del fill s_{i+1}-èsim d'x' (vegeu l'exemple), altrament la seqüència s no és vàlida. La seqüència buida $s = [\,]$ designa a l'arrel d'a, per tant és vàlida sempre que a no sigui buit.



Representarem les seqüències mitjançant cues (`queue`). Recordeu que el primer element d'una cua no buida q es pot consultar amb `q.front()` i es pot eliminar amb `q.pop()`.

(a) (2,5 punts) Implementa el següent mètode de classe privat:

```
// Pre: p ≠ nullptr i s = S i tots els elements d'S, si n'hi ha,
// són estrictament positius
// Post: retorna un punter al node designat per la seqüència donada a la
// cua s o nullptr si la seqüència no és vàlida, per a l'arbre general
// l'arrel del qual està apuntada per p; si S es vàlida llavors
// s = [] i si S no és vàlida el contingut d's pot ser qualsevol
static node_arbreGen* navega_fins_node(node_arbreGen* p, queue<int>& s);
```

(b) (1 punt) Implementa els següents mètodes consultors:

```
// Pre: s = S
// Post: retorna cert si i només si l'arbre implícit no és buit
// i la seqüència S donada és vàlida per a l'arbre implícit; a més,
// si S és vàlida llavors s = [] i si S no és vàlida el contingut
// d's pot ser qualsevol
bool es_valida(queue<int>& s) const;
```

```
// Pre: la seqüència s = S és vàlida per a l'arbre implícit
// Post: retorna el valor (l'atribut info)
// del node x designat per la seqüència S; a més, s = []
T valor(queue<int>& s) const;
```

(c) (1,5 punts) Implementa el següent mètode modificador:

```
// Pre: la seqüència s = S és vàlida per a l'arbre implícit i l'arbre implícit
// no és buit
// Post: si x és el node designat per S, elimina tots els descendents
// d'x; a més, el node x resta com una fulla de l'arbre implícit i s = []
void cut(queue<int>& s);
```

SOLUCIÓ:

Empleneu les caixetes buides del codi que us adjuntem

Cognoms, Nom **(Amb lletres majúscules)** D.N.I.

(a)

```
// Pre: p ≠ nullptr i s = S i tots els elements d'S, si n'hi ha,
// són estrictament positius
// Post: retorna un punter al node designat per la seqüència donada a la
// cua s o nullptr si la seqüència no és vàlida, per a l'arbre general
// l'arrel del qual està apuntada per p; si S es vàlida llavors
// s = [] i si S no és vàlida el contingut d's pot ser qualsevol
static node_arbreGen* navega_fins_node(node_arbreGen* p, queue<int>& s) {
  if (s.empty()) { return                    ; }
  if (                    ) {
        return nullptr;
  }

  int x =                    ;
  node_arbreGen* q =                    ;
  while (                    and x != 1) {
     ;
     ;
  }
  if (q == nullptr) {
    return                    ;
  } else {
     ;
    return                    ;
  }
}
```

(b)

```
// Pre: s = S
// Post: retorna cert si i només si l'arbre implícit no és buit
// i la seqüència S donada és vàlida per a l'arbre implícit; a més,
// si S és vàlida llavors s = [] i si S no és vàlida el contingut
// d's pot ser qualsevol
bool es_valida(queue<int>& s) const {
  return                          and                          ;
}


// Pre: la seqüència s = S és vàlida per a l'arbre implícit
// Post: retorna el valor (l'atribut info)
// del node x designat per la seqüència S; a més, s = []
T valor(queue<int>& s) const {
  // la primera capseta pot deixar-se buida
                                  ;
  return                          ;
}
```

(c)

```
// Pre: la seqüència s = S és vàlida per a l'arbre implícit i l'arbre implícit
// no és buit
// Post: si x és el node designat per S, elimina tots els descendents
// d'x; a més, el node x resta com una fulla de l'arbre implícit i s = []
void cut(queue<int>& s) {
   node_arbreGen* p =                          ;
   esborra_node_arbre(                         );
                                  ;
}

// Pre: cert
// Post: si q no és nullptr s'han esborrat tots els nodes accessibles
// des de q, inclòs; en un altre cas no es fa res
static void esborra_node_arbre(node_arbreGen* q) {
  if (                          ) {
                                ;
                                ;
                                ;
  }
}
```