Examen Programació 2

Grau en Informàtica
Juny 2011
Temps estimat: 2h 50m

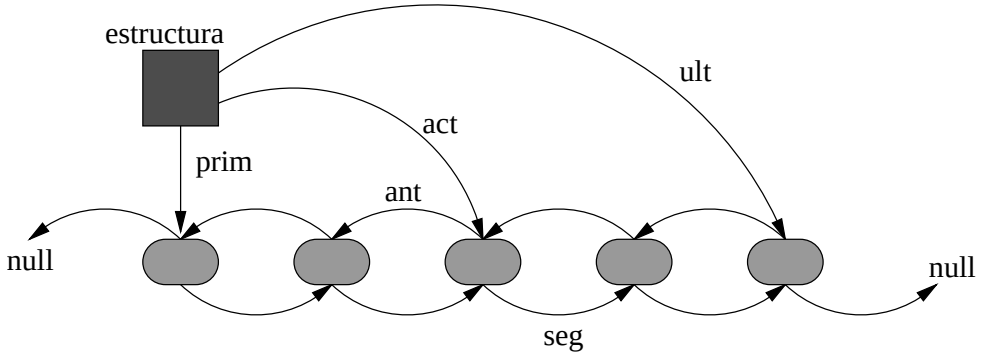
Problema 1: *Preu just i "splice" de llistes* (50%)
Suposem la següent representació de tipus basada en nodes d'una classe *Llista* per manegar llistes genèriques d'elements de tipus `T`:

```
template <class T> class Llista {
  private:
    struct node_llista {
        T info;
        node_llista* seg;
        node_llista* ant;
    };

    int longitud;
    node_llista* prim;
    node_llista* ult;
    node_llista* act;
    ...  // especificació i implementació d'operacions privades

  public:
    ...  // especificació i implementació d'operacions públiques
};
```

Els nodes són doblement encadenats amb punters al següent (`seg`) i a l'anterior (`ant`), i la llista té quatre atributs: la `longitud` i tres punters a nodes, un pel primer element (`prim`), un per l'últim (`ult`) i un altre per l'element actual (`act`), on tenim situat el punt d'interès de la llista. Gràficament, la implementació de l'estructura és



Noteu, per tant, que es tracta d'una implementació **sense** sentinella. Recordeu que l'element `act` d'una llista sense sentinella pot ser `null` fins i tot si la llista no és buida. És en aquest cas quan diem que el punt d'interès hi és a la dreta del tot.

Volem implementar dins d'aquesta classe dues operacions noves amb les següents especificacions pre/post:

```
void moure_punt_preu_just(const T &x)
/* Pre: el paràmetre implícit no té elements repetits */
/* Post: el punt d'interès del paràmetre implícit passa a estar sobre
   l'element més gran del paràmetre implícit que sigui estrictament menor
   que x, si n'hi ha; en cas contrari passa a estar a la dreta del tot */

void splice(Llista &l)
/* Pre: el paràmetre implícit = P, l = L */
/* Post: el paràmetre implícit conté el resultat d'inserir la llista L
   en la llista P a l'esquerra del punt d'interès de P; el punt d'interès
   del paràmetre implícit no canvia; l és una llista buida */
```

Exemples:

1. `moure_punt_preu_just`: si $x = 5$ i el paràmetre implícit és

   ```
   -1    3    -7    14    -2    5    -6    9    1    4    2
   ```

   el nou punt d'interès ha de quedar sobre el 4.

2. `splice`: si el paràmetre implícit és

   ```
   1    2    3    4    5
   ```

   amb el punt d'interès sobre el 3 i la llista `l` és

   ```
   6    7    8    9
   ```

   llavors el nou paràmetre implícit ha de ser

   ```
   1    2    6    7    8    9    3    4    5
   ```

   amb el punt d'interès sobre el 3 i `l` ha de quedar buida.

Es demana dissenyar implementacions d'aquestes operacions que **no usin cap operació primitiva de les llistes** (per això no les recordem en aquest enunciat) **ni iteradors**, sinó que accedeixin directament als atributs de la classe *Llista*.

No s'han de justificar, però podeu incloure comentaris aclaridors si ho creieu convenient.

Es valorarà l'eficiència de les solucions proposades.

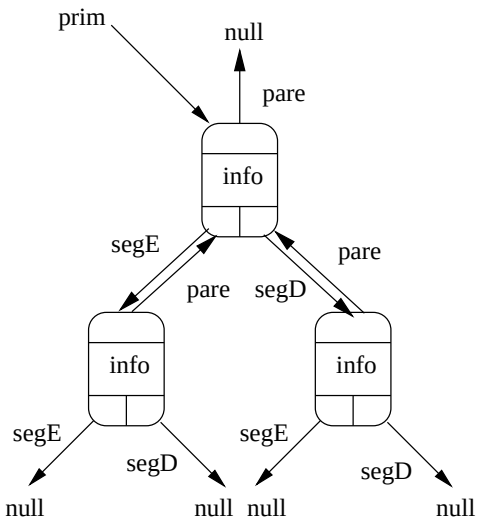
Problema 2: *Arbres binaris amb punter al pare* (50%)
Una possible extensió de la classe *Arbre* que hem vist pels arbres binaris consisteix en afegir operacions per saber si un arbre té pare (és a dir, si és fill esquerre o dret d'un altre arbre) i, cas que en tingui, obtenir el seu arbre pare. Anomenarem aquesta nova classe *ArbrePlus* i proposem per ella la següent representació basada en nodes:

```
template <class T> class ArbrePlus {
  private:
    struct node_arbrePlus {
        T info;
        node_arbrePlus* segE;
        node_arbrePlus* segD;
        node_arbrePlus* pare;
    };

    node_arbrePlus* prim;
    ...  // especificació i implementació d'operacions privades

  public:
    ...  // especificació i implementació d'operacions públiques
};
```

Els nodes tenen punters al primer node del fill esquerre (`segE`), al primer node del fill dret (`segD`) i a l'arrel del pare (`pare`), si en tenen; si no, el seu valor és `null` a qualsevol dels tres casos. L'arbre només té un atribut: un punter al primer node (`prim`). Gràficament, la implementació de l'estructura és



Us demanem implementar dues operacions d'aquesta classe, la constructora de còpia i la de plantar un nou arbre, amb les següents especificacions pre/post:

```
ArbrePlus(const ArbrePlus &orig)
/* Pre: cert */
/* Post: el resultat és una còpia de l'arbre orig */

void plantar(const T &x, ArbrePlus &a1, ArbrePlus &a2)
/* Pre: el paràmetre implícit és buit, a1=A1, a2=A2 */
/* Post: el paràmetre implícit té x com a arrel, A1 com a fill
         esquerre i A2 com a fill dret i, per tant, passa a ser
         el pare dels arbres A1 i A2; a1 i a2 són buits */
```

La constructora `ArbrePlus` només haurà de cridar una operació privada recursiva, que també heu de dissenyar a partir de la següent especificació:

```
node_arbrePlus* copia_node_arbrePlus(node_arbrePlus* m)
/* Pre: cert */
/* Post: el resultat és NULL si m és NULL; en cas contrari, el
         resultat apunta al node arrel d'una jeraquia de nodes
         que és una còpia de la jerarquia de nodes que té el node
         apuntat per m com a arrel */
```

Es demana dissenyar implementacions d'aquestes operacions que **no usin cap operació primitiva dels arbres** (per això no les recordem en aquest enunciat) sinó que accedeixin directament als atributs de la classe *ArbrePlus*.

No s'han de justificar, però podeu incloure comentaris aclaridors si ho creieu convenient.

Es valorarà l'eficiència de les solucions proposades.