Curso 2021-22 Q1, 2º Parcial

1. Suavización de una lista

Se pide implementar la siguiente función sobre listas con doble encadenamiento, sin centinela.

```
/* Pre:  L.size()≥2 */
/* Post: S'ha afegit a la llista ímplicita un nou
element al mig dels dos elements consecutius que
estan a una distància de valor més gran (en cas
d'empat, els que siguin més a prop de l'inici de
la llista); el valor del nou element és la (part
entera de la) mitjana del valor d'aquests dos
elements veins; el punt d'interès de la llista
resultant passa a apuntar al nou element afegit */
void suaviza ();
```

```
void suaviza (){
   nodo_lista* ant = primer_nodo;
   nodo_lista* antseg = ant -> sig;
   int dist_max = abs(antseg->info - ant->info);
   nodo_lista* prim_dmax = ant;
   ant = ant->seg; antseg = ant -> sig;
   /* Inv: si L es la lista implícita, distmax es
la dist. màxima de dos elements consecutivos de
la subllista de L entre el primer_nodo y ant, y
prim_dist apunta al primero, antseg = ant->seg */
   while (antseg != nullptr){
      int dist = abs(antseg->info - ant->info);
      if(dist > dist_max) {
          dist_max = dist;
          primdmax = ant;
      }
      ant = antseg;
      antseg = ant -> sig;
   }
   ...
```

Suavización de una lista

```
    nodo_lista* seg_dmax = prim_dmax->seg;
    act = new nodo_lista;
    act->info = (prim_dmax->info+seg_dmax->info)/2;
    prim_dmax->seg =act;
    act->ant = prim_dmax;
    act->seg = seg_dmax;
    seg_dmax->ant = act;
    ++longitud;
}
```

2. Construcción de un árbol lleno

Se pide implementar la siguiente función sobre árboles binarios

```
/* Pre:  L'arbre implícit és buit i k >= 0 */
/* Post: L'arbre implícit és un arbre binari amb k
nivells plens de nodes, on cada node té com a
valor la seva profunditat a l'arbre */
void constr_lleno (int k);
```

Para implementar esta función se ha de usar la siguiente inmersión:

```
/* Pre:  k >= 0, 0 <= prof <= k */
/* Post: retorna un apuntador al node arrel
(si existeix) d'un arbre binari amb (k-prof)
nivells plens de nodes, on cada node té com a
valor la seva profunditat a l'arbre + prof*/
static nodo_arbol i_constr_lleno (int k, int
prof);
```

```
void constr_lleno (int k){
   primer_nodo = i_constr_lleno(k,0);
}

static nodo_arbol i_constr_lleno (int k, int
prof){
   if (prof == k) return nullptr;
   else {
      nodo_arbol* p = new nodo_arbol;
      p->info = prof;
      p->segI = i_constr_lleno (k, prof + 1);
      p->segD = i_constr_lleno (k, prof + 1);
      return p;
   }
}
```

Curso 2020-21 Q2, 2º Parcial

1. Eliminación de pares

Se pide implementar las siguientes funciones sobre listas con doble encadenamiento, sin centinela.

```
/* Pre:  La llista implícita conté almenys dos
elements; el punt d'interès apunta un element de
la llista que té un successor (és a dir, no apunta
l'últim) */
/* Post: S'ha eliminat de la llista ímplicita el
punt d'interès i el seu successor; a la llista
resultant el punt d'interès passa a ser el
successor del successor del punt d'interès
original */
void elimina2 ();
```

```
/* Pre: La llista implícita és una llista L no
buida */
/* Post: La llista ímplicita és el resultar
d'eliminar d'L tots els parells d'elements
consecutius que sigui necessari de manera que no
hi hagi cap parell d'elements consecutius la suma
dels quals sigui 0 */
void elimina_par_0 ();
```

```
void elimina2 (){
   nodo_lista* aux = act;
   act = act -> seg -> seg;
   if (longitud == 2){
      primero = ultimo = nullptr;
   } else if (primero == aux)
      primero = act;
      act->ant = nullptr;
   } else if (aux->sig == ultimo){
      ultimo = aux->ant;
      ultimo->sig = nullptr;
   } else {
      act->ant=aux->ant;
      (aux->ant)->sig = act;
   }
   delete aux->sig; delete aux;
   longitud-= 2;
}
```

```
void elimina_par_0 (){
    act = primero;
    nodo_lista* p = primero->sig;
/* No hay dos nodos consecutivos que sumen 0 antes
del nodo apuntado por p, si act != nullptr,
entonces p == act->sig */
    while (p != nullptr) {
        if (p->info + act->info == 0) {
            elimina2();
            if (act == nullptr) p = nullptr;
            else if (act = prmero) p = act.sig;
            else { p = act; act = p->ant; }
        } else {
            act = p;
            p = p->sig;
        }
    }
}
```

2. Árbol de múltiplos

Se pide implementar las siguientes funciones sobre árboles N-arios:

1. Un método privado es_fulla que retorne cierto si y solamente si el nodo apuntado por un puntero p es una hoja (tiene todos sus hijos vacíos.
2. Un método público mult que retorne cierto si y solo si todo subárbol del parámetro implícito cumple que la suma de los valores del subárbol es un múltiplo del número de nodos del subárbol.

```
/* Pre:  Cert */
/* Post: retorna true si i només si p apunta a
l'arrel d'un arbre N-ari consistent en un únic
node*/
static bool es_hoja (nodo* p){
   if (p == nullptr) return false;
   for (int i = 0; i < p->hijos.size(); ++i)
      if (p->hijos[i] != nullptr) return false;

   return true;
}
```

```
/* Pre: l'arbre N-ari implícit no és buit*/
/* Post:  retorna true si i només si tots els
subarbres de l'arbre implícit compleixen que la
suma dels seus valors és múltiple del seu nombre
de nodes*/
bool mult ();
```

Para implementar mult es necesario implementar la siguiente inmersión:

```
/* Pre:  p és un punter a l'arrel d'un arbre N-ari
no buit*/
/* Post: retorna true si i només si tots els
subarbres de l'arbre de amb node arrel apuntat per p
compleixen que la suma dels seus valors és múltiple
del seu nombre de nodes. A més, si la funció retorna
true, llavors s és la suma de tots els valor del
arbre apuntat per p i n la seva talla */
static bool i_mult (nodo* p, int& s, int& n);
```

```
bool mult () {
    nodo* p = primero;
    int s,n;
    return i_mult (p, s, n);
}
static bool i_mult (nodo* p, int& s, int& n){
    if (es_fulla(p)){
        s = p->info; n = 1; return true;
    } else {
        bool cumple = true;
        s = p->info; n = 1;
        for (int = 0;i<p->hijos.size() and cumple;++i}{
            if(p ->hijos[i] != nullptr){
                int si,ni;
                cumple = i_mult(p->hijos[i],si,ni);
                s += si; n +=ni
        }   }
        if (cumple) return s % n == 0
        else return false;
}
```