# *Exàmens de PRG - Problemes del Tema 5*
# *Estructures de dades lineals*

## Curs 2018/19

**P2 - Curs 18/19:** 3.5 punts

**Es demana:** afegir un mètode a la classe `QueueIntLinked` amb perfil:

```
public void split(int x)
```

tal que, donat un enter `x`, cerque la primera ocurrència de l'element `x` en la cua i el substituïsca pel parell d'elements `x / 2` i `x / 2 + x % 2`, un a continuació de l'altre. Si `x` no apareix, la cua no ha de canviar.

Per exemple, si s'invoca al mètode `q.split(9)` sent `q` la cua de longitud 6 ← `1 -2 9 8 -3 5` ←, llavors `q` passa a ser la cua de longitud 7 ← `1 -2 4 5 8 -3 5` ←.

**IMPORTANT:** En la solució només es pot accedir als atributs de la classe, quedant prohibit accedir als seus mètodes.

**Solució:**

```
public void split(int x) {
     NodeInt aux = this.first;
     while (aux != null && aux.data != x) {
        aux = aux.next;
     }
     if (aux != null) {
         aux.data = x / 2;
         aux.next = new NodeInt( x / 2 + x % 2, aux.next);
         if (aux == this.last) { this.last = aux.next; }
         this.size++;
     }
}
```

**P2 - Curs 18/19:** 3.5 punts

**Es demana:** implementar un mètode estàtic `compress` tal que, donada una `ListPIIntLinked l` de la qual se suposa que els seus elements valen tots 0 o 1, retorne una altra llista de grandària aproximadament la meitat, tal que els seus elements representen als de `l`, prenent-los de dos en dos i d'esquerra a dreta. Així, on en `l` apareix una parella d'elements seguits `e1 e2`, en la nova llista apareix `e1 * 2 + e2` (un valor `0`, `1`, `2`, o `3` per a les parelles `00`, `01`, `10`, `11`, respectivament). En el cas en què en `l` quedara al final un element `e` desemparellat, en la nova llista apareixeria al final `e - 2` (un valor `-1` o `-2`). Els elements de la llista `l` no han de canviar, encara que la posició del punt d'interés sí que pot canviar.

Per exemple, si `l` és una llista amb els elements `0 0 0 1 1 0 1 1 0 0 1`, el mètode ha de retornar una altra llista amb els elements `0 1 2 3 0 -1`.

**IMPORTANT:** Se suposarà que el mètode s'implementa en una classe diferent a `ListPIIntLinked`, per tant, només es podran usar els mètodes públics de la classe.

**Solució:**

```
/** Precondició: els elements de l valen 0 o 1. */
public static ListPIIntLinked compress(ListPIIntLinked l) {
    ListPIIntLinked result = new ListPIIntLinked();
    int n = l.size();
    l.begin();
    while (n >= 2) {
        int e1 = l.get(); l.next();
```

```
            int e2 = l.get(); l.next();
            result.insert(e1 * 2 + e2);
            n = n - 2;
        }
        if (n == 1) { result.insert(l.get() - 2); }
        return result;
    }
```

**RecP2 - Curs 18/19:** 3.5 punts

**Es demana:** implementar un mètode d'instància en la classe `ListPIIntLinked` amb el perfil i la precondició següents:

```
/** Precondició: la llista this conté dades emmagatzemades en ordre creixent */
public void removeGreaterThan(int e)
```

Donat un enter `e`, el mètode modifica la llista esborrant de la mateixa tots els elements majors que `e`. Al final de l'execució, el `PI` ha d'estar a l'inici de la llista.

Per exemple, si la llista `l` inicialment és `[10] 12 14 15`, i l'enter `e` és `12`, aleshores, la llista, després d'executar el mètode, quedarà com `[10] 12`. Considerant la mateixa llista `l`, i l'enter `e` = `9`, aleshores, la llista es quedarà buida després d'executar el mètode.

**REQUISIT:** Utilitzar solament els atributs de la classe i referències auxiliars a `NodeInt`, no als mètodes de la classe.

**Solució:**

```
    public void removeGreaterThan(int e) {
        if (first != null && first.data > e) {
            first = null;
            size = 0;
        }
        else {
            int cont = 0;
            NodeInt aux = first, prev = null;
            while (aux != null && aux.data <= e) {
                prev = aux;
                aux = aux.next;
                cont++;
            }
            if (aux != null) {
                prev.next = null;
                size = cont;
            }
        }
        prevPI = null;
        pI = first;
    }
```

**RecP2 - Curs 18/19:** 3.5 punts

**Es demana:** implementar un mètode estàtic amb perfil

```
public static void moureAlFinal(QueueIntLinked q, int x)
```

que cerque la primera ocurrència de l'element `x` dins de la cua `q`, i,

- en cas de trobar-lo, el lleve d'on està i el pose com el darrer element de la cua.
- En cas contrari, deixa la cua intacta.

**REQUISIT:** Es suposarà que el mètode s'implementa en una classe diferent a `QueueIntLinked`, aleshores, sols es podrà fer ús dels mètodes públics de la classe `QueueIntLinked`.

**Solució:**

```
public static void moureAlFinal(QueueIntLinked q, int x) {
    int n = q.size(), i = 0;
    while (i < n && q.element() != x) {
        q.add(q.remove());
        i++;
    }
    if (i < n) {
        q.remove();
        for (int j = i + 1; j < n; j++) {
            q.add(q.remove());
        }
        q.add(x);
    }
}
```

# Curs 2017/18

**P2 - Curs 17/18:** 2.5 punts

**Es demana:** implementar un mètode estàtic tal que, donat un `String s`, el convertisca en una seqüència enllaçada de caràcters on el primer element de la seqüència ha de ser el primer caràcter de l'`String s`. Si l'`String s` està buit o és `null`, la seqüència resultant serà també `null`; si no, retornarà el primer node de la seqüència. Per a això, es suposa accessible una classe `NodeChar` idèntica a la clase `NodeInt` utilitzada en el paquet `linear`, excepte en el tipus de la dada.

Per exemple, donat l'`String s = "Examen"`, la seqüència serà: → 'E' → 'x' → 'a' → 'm' → 'e' → 'n'

**Solució:**

```
public static NodeChar fromStringToSeq(String s) {
    NodeChar res = null;
    if (s != null) {
        for (int i = s.length() - 1; i >= 0; i--) {
            char c = s.charAt(i);
            res = new NodeChar(c, res);
        }
    }
    return res;
}
```

**P2 - Curs 17/18:** 2.5 punts

**Es demana:** Afegir un mètode a la classe `ListPIIntLinked` amb el perfil:

```
public void append(int x)
```

tal que, donat un enter `x`, l'inserisca en la posició següent al punt d'interès, avançant aquest al nou element introduït. Si el cursor ja està al final en el moment d'invocar al mètode, aquest ha de llançar l'excepció `NoSuchElementException` amb el missatge `"Cursor al final"`.

Per exemple, si s'invoca al mètode `l.append(5)` sent `l` la llista (on l'element distingit és el marcat entre claudàtors) `1 4 [7] 8 3 4` , aleshores `l` queda de la forma `1 4 7 [5] 8 3 4` .

**IMPORTANT:** En la solució només es permet accedir als atributs de la classe, quedant prohibit l'accés als seus mètodes.

**Solució:**

```
public void append(int x) {
    if (pI != null) {
        prevPI = pI;
        pI.next = new NodeInt(x, pI.next);
        pI = pI.next;
        size++;
    } else { throw new NoSuchElementException("Cursor al final"); }
}
```

**P2 - Curs 17/18:** 2.5 punts

**Es demana:** implementar un mètode estàtic `fusion` que, donades dues cues `QueueIntLinked q1` i `q2`, retorne una nova cua en la que s'hagen fusionat els elements de `q1` i `q2` de manera que apareguen per ordre de cua, però alternant-se un element d'una i altra cua. Els elements sobrants d'alguna de les cues apareixeran al final de la cua resultat. Les cues `q1` i `q2` han de quedar en el seu estat original.

Per exemple, si `q1` és ← `3 6 20 1 -3 4 -5` ← i `q2` és ← `10 9 8` ←
el mètode retorna la cua ← `3 10 6 9 20 8 1 -3 4 -5` ←.

**IMPORTANT:** Es suposarà que el mètode s'implementa en una classe distinta de `QueueIntLinked`, per tant, només es podran usar els mètodes públics de la classe.

**Solució:**

```
public static QueueIntLinked fusion(QueueIntLinked q1, QueueIntLinked q2) {
    QueueIntLinked res = new QueueIntLinked();
    int i = Math.min(q1.size(), q2.size());
    for (int j = 0; j < i; j++) {
        res.add(q1.element()); q1.add(q1.remove());
        res.add(q2.element()); q2.add(q2.remove());
    }
    while (i < q1.size()) { res.add(q1.element()); q1.add(q1.remove()); i++; }
    while (i < q2.size()) { res.add(q2.element()); q2.add(q2.remove()); i++; }
    return res;
}
```

**RecP2 - Curs 17/18:** 2.5 punts

**Es demana:** implementar un mètode estàtic iteratiu que copie els elements de `l`, una llista `ListPIIntLinked` passada com a paràmetre, en una nova llista, resultat a retornar, els elements de la qual estiguen en el mateix ordre, però restant a tots els elements el valor mínim de la llista `l`. Per exemple, si la llista original `l` és `[12] 50 10 120` , el mínim és `10` i, per tant, s'ha d'obtindre la llista `2 40 0 110 [ ]` .

A més, cal tindre en compte que:

- Si la llista `l` està buida, s'ha de retornar `null`.
- En qualsevol cas, el contingut inicial de la llista `l` s'ha de conservar, a excepció de la posició del seu punt d'interés (que es permet modificar).
- **REQUISIT:** El mètode demanat és d'una classe diferent a `ListPIIntLinked`. Per tant, la seua implementació ha de fer-se usant els mètodes públics de `ListPIIntLinked` exclusivament.

**Solució:**

```
public static ListPIIntLinked subtractMinimumToList(ListPIIntLinked l) {
    if (l.empty()) { return null; }
    ListPIIntLinked res = new ListPIIntLinked();
    l.begin();
    int min = l.get();
    while (!l.isEnd()) {
        if (l.get() < min) { min = l.get(); }
        l.next();
    }
    for (l.begin(); !l.isEnd(); l.next()) {
        res.insert(l.get() - min);
    }
    return res;
}
```

**RecP2 - Curs 17/18:** 2.5 punts

**Es demana:** implementar un mètode estàtic que reba una seqüència enllaçada `NodeInt` i retorne una altra seqüència enllaçada `NodeInt` que continga solament les dades parelles de la seqüència rebuda. Per exemple, si la seqüència rebuda conté les següents dades: `4 7 2 8 9 3 6`, s'ha de retornar la següent seqüència: `4 2 8 6`. S'ha de tindre en compte que si la seqüència rebuda és `null` o si no conté dades parelles, s'ha de retornar `null`.

**Solució:**

```
public static NodeInt evenSubsequence(NodeInt seq) {
    NodeInt first = null, last = null;
    while (seq != null) {
```

```
            if (seq.data % 2 == 0) {
                if (first == null) {
                    last = new NodeInt(seq.data);
                    first = last;
                }
                else {
                    last.next = new NodeInt(seq.data);
                    last = last.next;
                }
            }
            seq = seq.next;
        }
        return first;
    }
```

**RecP2 - Curs 17/18:** 2.5 punts

**Es demana:** implementar en la classe `QueueIntLinked` un mètode d'instància que divideixca una cua en dues mitats, amb perfil `public QueueIntLinked divideQueue()`.

Tot i tenint en compte que:

- Precondició: la cua inicial (`this`) té almenys dos elements.
- La divisió es realitza de forma que la cua inicial es queda amb la primera mitat dels elements, i es retorna una cua amb la resta dels elements.
- Les cues resultants mantenen l'ordre dels elements en la cua inicial.
- Si la cua inicial té un quantitat imparell d'elements, serà la cua retornada la que tindrà una longitud superior en una unitat.

Exemples:

| Cua inicial | Cua inicial modificada | Cua retornada |
|---|---|---|
| 1 2 2 4 3 1 | 1 2 2 | 4 3 1 |
| 1 2 2 4 3 1 1 | 1 2 2 | 4 3 1 1 |

**REQUISIT:** En el mètode demanat cal usar exclusivament els atributs de `QueueIntLinked`, el seu constructor, i referències a `NodeInt`. Per tant, NO es permet cap invocació als mètodes de la classe.

**Solució:**

```
    public QueueIntLinked divideQueue() {
        QueueIntLinked nq = new QueueIntLinked();
        int middle = size / 2;
        NodeInt aux = this.first;
        for (int i = 0; i < middle - 1; i++) {
            aux = aux.next;
        }
        nq.first = aux.next;
        nq.last = this.last;
        aux.next = null;
        this.last = aux;
        nq.size = this.size - middle;
        this.size = middle;
        return nq;
    }
```

# Curs 2016/17

**P2 - Curs 16/17:** 2.5 punts

**Es demana:** implementar un nou constructor en la classe `StackIntLinked`, tal que donada una `StackIntLinked` `p` torne una altra pila que siga una còpia independent de `p`. Així, si s'executa la instrucció:

```
StackIntLinked res = new StackIntLinked(StackIntLinked p);
```

es podrà empilar i desempilar de la pila `res` o de la pila `p` sense influir una a l'altra.

**NOTA:** Només es permet accedir als atributs de la classe, quedant prohibit l'accés als seus mètodes.

**Solució:**

```
public StackIntLinked(StackIntLinked p) {
    if (p.top != null) {
        top = new NodeInt(p.top.data);
        NodeInt last = top;
        NodeInt aux = p.top.next;
        while (aux != null) {
            last.next = new NodeInt(aux.data);
            last = last.next;
            aux = aux.next;
        }
        size = p.size;
     }
}
```

**P2 - Curs 16/17:** 2.5 punts

Es vol implementar el mètode `equals(Object)` a la classe `ListPIIntLinked`, per definir quan la `ListPIIntLinked` sobre la qual s'aplique aquest mètode és o no igual a una altra que es rebrà com a argument.

Donades dues `ListPIIntLinked` qualsevol, es consideren iguals quan:

1. Tenen els mateixos elements en les mateixes posicions i
2. el seu *Punt d'Interès* és el mateix, és a dir, es troba en la mateixa posició en les dues llistes.

**Es demana:** completar el codi següent perquè el mètode `equals(Object)` s'execute segons la definició anterior:

```
/** Retorna si l'objecte en curs és igual a o */
public boolean equals(Object o) {
    boolean res = true;
    if (!(o instanceof ListPIIntLinked)) { res = false; }
    else {
        ListPIIntLinked altra = (ListPIIntLinked) o;

        // COMPLETA el codi per tornar si "this" i "altra" són iguals
}
```

**NOTA:** La posició del *Punt d'Interès* haurà de romandre inalterada després de l'execució del mètode.

**Solució:**

```
/** Retorna si l'objecte en curs és igual a o */
public boolean equals(Object o) {
    boolean res = true;
    if (!(o instanceof ListPIIntLinked)) { res = false; }
    else {
        ListPIIntLinked altra = (ListPIIntLinked) o;
        if (this.size != altra.size) { res = false; }
```

```
            else {
                NodeInt p = this.first, q = altra.first;
                while (p != null && res) {
                    res = (p.data == q.data);
                    if (res && p == this.prevPI) { res = (q == altra.prevPI); }
                    p = p.next; q = q.next;
                }
            }
        }
        return res;
    }
```

**P2 - Curs 16/17:** 2.5 punts

Per problemes de codificació, s'ha rebut cert text en què totes les lletres `'ç'` han estat canviades pel parell de caràcters `'\\c'` i totes les `'Ç'` pel parell `'\\C'`.

Es disposa d'una classe ja creada, `ListPICharLinked`, que implementa una llista amb punt d'interès, els elements de la qual són caràcters, i amb els següents mètodes públics:

```
public void begin()           public void next()
public void insert(char e)    public char remove()
public char get()             public int size()
public boolean empty()        public boolean isEnd()
```

**Es demana:** implementar un mètode estàtic, extern a la classe `ListPICharLinked`, que donat un objecte d'aquest tipus que conté, caràcter a caràcter, cert text amb el problema de codificació anteriorment exposat, modifique el contingut d'aquesta llista per codificar-la de forma correcta.

**Per exemple:** donada una llista amb el text `"En PU\\COL, una on\\ca de xocolata es un bon dol\\c."`, s'haurà de modificar perquè quede `"En PUÇOL, una onça de xocolata es un bon dolç"`.

**NOTA:** Considereu que si apareix el caràcter `'\\'`, aquest sempre anirà seguit d'una `'c'` o una `'C'`.

**Solució:**

```
    public static void modifTrencada(ListPICharLinked l) {
        l.begin();
        while (!l.isEnd()) {
            char c1 = l.get();
            if (c1 == '\\') {
                l.remove();
                char c2 = l.remove();
                if (c2 == 'c') { l.insert('ç'); }
                else { l.insert('Ç'); }
            } else { l.next(); }
        }
    }
```

**RecP2 - Curs 16/17:** 2.5 punts

Es disposa de la següent classe que implementa una seqüència enllaçada amb nodes de tipus `NodeInt`:

```
public class SecEnla {
    private NodeInt sec;  // referència al primer element de la seqüència
    . . .
}
```

**Es demana:** implementar un mètode d'instància a la classe `SecEnla` que, donat un valor enter `x`, trobe la primera ocurrència d'aquest valor en la seqüència i l'avance una posició. Si `x` es troba en el primer node de la

seqüència, és a dir, no es pot avançar una posició, passarà a l'últim i viceversa. En cas que `x` no es trobe, no farà res. Se suposa, per precondició, que la seqüència té, almenys, 2 nodes.

Per exemple, si la seqüència conté `[3, 7, 2, 8, 5]` i executem la crida al mètode `avançar(8)`, la seqüència quedarà `[3, 7, 8, 2, 5]`. Si, a continuació, executem `avançar(3)`, la seqüència quedarà `[5, 7, 8, 2, 3]`.

**Solució:**

```
/** Precondició: la seqüència té, almenys, 2 nodes */
public void avançar(int x) {
    NodeInt ant = null, aux = sec;
    while (aux != null && aux.data != x) {
        ant = aux;
        aux = aux.next;
    }
    if (aux != null) {
        if (ant == null) {
            ant = aux;
            while (ant.next != null) { ant = ant.next; }
        }
        aux.data = ant.data;
        ant.data = x;
    }
}
```

**RecP2 - Curs 16/17:** 2.5 punts

**Es demana:** implementar un mètode d'instància a la classe `ListPIIntLinked` amb perfil

```
public void inserirAl(int x, boolean inici)
```

que, donat un enter `x`, l'inserisca a l'inici de la llista si el paràmetre `inici` és `true` i, en cas contrari, l'inserisca al final. El punt d'interès ha de quedar sobre l'element inserit.

**Nota:** Només es permet accedir als atributs de la classe, quedant prohibit l'accés als seus mètodes.

**Solució:**

```
public void inserirAl(int x, boolean inici) {
    NodeInt nou = new NodeInt(x);
    if (first == null) { first = nou; }
    else if (inici) {
        nou.next = first;
        first = nou;
        prevPI = null;
    }
    else {
        NodeInt aux = first;
        while (aux.next != null) { aux = aux.next; }
        aux.next = nou;
        prevPI = aux;
    }
    pI = nou;
    size++;
}
```

**RecP2 - Curs 16/17:** 2.5 punts

**Es demana:** implementar un mètode estàtic fora de la classe `QueueIntLinked` tal que, donada una `QueueIntLinked` amb valors en [0..9], retorne el nombre enter format pels dígits de la mateixa. La cua ha de quedar en el seu estat original. Per exemple, si la cua és ← `5 1 4 7` ←, l'enter resultant serà `5147`. Fixa't que aquest enter pot obtenir-se com segueix: `(((((5 * 10) + 1) * 10) + 4) * 10) + 7`.

**Solució:**

```
/** Versió 1: sense estructures auxiliars */
public static int fromCuaToInt(QueueIntLinked q) {
    int res = 0;
    for (int i = 0; i < q.size(); i++) {
        int x = q.remove();
        res = res * 10 + x;
        q.add(x);
    }
    return res;
}

/** Versió 2: usant una cua auxiliar */
public static int fromCuaToInt(QueueIntLinked q) {
    int res = 0;
    QueueIntLinked qAux = new QueueIntLinked();
    while (!q.empty()) {
        int x = q.remove();
        res = res * 10 + x;
        qAux.add(x);
    }
    while (!qAux.empty()) {
        int x = qAux.remove();
        q.add(x);
    }
    return res;
}
```

# Curs 2015/16

**P2 - Curs 15/16:** 3 punts

En una cua, de vegades, pot ser d'interès treure algun element `x` indesitjat. Per aquest motiu, **es demana** afegir a la classe `QueueIntLinked` un nou mètode amb perfil:

```
public int remove(int x)
```

que traga de la cua la primera ocurrència de `x`, i la retorne. En el cas en què la cua estiga buida ha de llançar l'excepció `NoSuchElementException` amb el missatge `Cua buida`, i si `x` no es troba ha de llançar una excepció del mateix tipus amb el missatge `x no es troba a la cua`.

**Nota:** a la solució no es podran fer servir els mètodes de la classe `QueueIntLinked`.

**Solució:**

```
public int remove(int x) {
    if (this.size == 0) { throw new NoSuchElementException("Cua buida"); }
    NodeInt ant = null, aux = this.first;
    while (aux != null && aux.data != x) {
        ant = aux; aux = aux.next;
    }
    if (aux != null) {
        if (this.first == aux) { this.first = aux.next; }
        else { ant.next = aux.next; }
        if (aux == this.last) { this.last = ant; }
        this.size--;
        return x;
    } else { throw new NoSuchElementException(x + " no es troba a la cua"); }
}
```

**P2 - Curs 15/16:** 3 punts

En una classe diferent a `ListPIIntLinked`, **es demana** implementar un mètode amb el següent perfil i precondició:

```
/** Precondició: llista1 i llista2 no contenen elements repetits. */
public static ListPIIntLinked elimComuns(ListPIIntLinked llista1, ListPIIntLinked llista2)
```

que retorne una llista amb els elements comuns d'ambdues llistes, eliminant de `llista1` aquests elements comuns.

**Exemple:**
Siga una `llista1` amb els valors `6 -5 4 8 -9`,
siga una `llista2` amb els valors `21 8 5 -9 -5 16`,
llavors el resultat de `elimComuns(llista1, llista2)` haurà de ser una llista `-5 8 -9`, i haurà de deixar `llista1` amb els elements `6 4`.

**Solució:**

```
/** Precondició: llista1 i llista2 no contenen elements repetits. */
public static ListaPIIntEnla elimComuns(ListPIIntLinked llista1, ListPIIntLinked llista2) {
    ListPIIntLinked result = new ListPIIntLinked();
    llista1.begin();
    while (!llista1.isEnd()) {
        int x = llista1.get();
        llista2.begin();
        while (!llista2.isEnd() && x != llista2.get()) { llista2.next(); }
        if (llista2.isEnd()) { llista1.next(); }
        else { llista1.remove(); result.insert(x); }
    }
    return result;
}
```

**RecP2 - Curs 15/16:** 3 punts

**Es demana:** afegir a la classe `QueueIntLinked` un mètode de perfil

```
public void recular(int x)
```

tal que:

- Busque la primera ocurrència de l'element `x` dins de la cua i, en cas d'èxit en la cerca, faça que aquest element es trasllade al final del tot i, per tant, es quede com l'últim de la cua.
- En cas de fracàs en la cerca, la cua es queda com estava.

**Nota:** Només es permet accedir als atributs de la classe, quedant totalment prohibit l'accés als seus mètodes, així com a qualsevol altra estructura de dades auxiliar (incloent l'ús d'arrays).

**Solució:**

```
/** Si x està en la cua, el fica l'últim de la cua. */
public void recular(int x) {
    NodeInt aux = first, ant = null;
    while (aux != null && aux.data != x) {
        ant = aux;
        aux = aux.next;
    }
    if (aux != null && aux != last) {
        if (aux == first) { first = first.next; }
        else { ant.next = aux.next; }
        last.next = aux;
        aux.next = null;
        last = aux;
    }
}
```

**RecP2 - Curs 15/16:** 3 punts

En una classe distinta a `ListPIIntLinked`, **es demana:** implementar un mètode amb el següent perfil i precondició:

```
/** Precondició: llista1 i llista2 no contenen elements repetits. */
public static ListPIIntLinked diferencia(ListPIIntLinked llista1, ListPIIntLinked llista2)
```

que torne una llista amb els elements de `llista1` que no estan en `llista2`.

Per exemple, donades la `llista1` $\rightarrow 7 \rightarrow 3 \rightarrow 9 \rightarrow 6 \rightarrow 2$ i la `llista2` $\rightarrow 8 \rightarrow 9 \rightarrow 5 \rightarrow 3 \rightarrow 2 \rightarrow 4$, aleshores el resultat de `diferencia(llista1, llista2)` ha de ser una llista amb els elements $\rightarrow 7 \rightarrow 6$.

**Solució:**

```
/** Precondició: llista1 i llista2 no contenen elements repetits. */
public static ListPIIntLinked diferencia(ListPIIntLinked llista1, ListPIIntLinked llista2) {
    ListPIIntLinked result = new ListPIIntLinked();
    llista1.begin();
    while (!llista1.isEnd()) {
        int x = llista1.get();
        llista2.begin();
        while (!llista2.isEnd() && x != llista2.get()) { llista2.next(); }
        if (llista2.isEnd()) { result.insert(x); }
        llista1.next();
    }
    return result;
}
```

# Curs 2014/15

**P2 - Curs 14/15:** 3 punts

**Es demana:** afegir a la classe `QueueIntLinked` un mètode amb perfil:

```
public void colar(int x)
```

tal que:

- Cerque la primera ocurrència de l'element `x` dins de la cua i en cas d'èxit en la cerca, faça que aquest element es "cole" davant del tot i, per tant, es quede com el primer de la cua.
- En cas de fracàs en la cerca, la cua es queda com estava.

**Nota:** Només es permet accedir als atributs de la classe, quedant totalment prohibit l'accés als seus mètodes.

**Solució:**

```
/** Si x està en la cua, el fica el primer en la cua. */
public void colar(int x) {
    NodeInt aux = first, ant = null;
    while (aux != null && aux.data != x) {
        ant = aux;
        aux = aux.next;
    }
    if (aux != null && aux != first) {
        ant.next = aux.next;
        aux.next = first;
        first = aux;
        if (aux == last) { last = ant; }
   }
}
```

**P2 - Curs 14/15:** 2 punts

Considereu la classe `ListPIIntLinked`, amb tots els mètodes coneguts i, a més, el mètode seguent, que es suposa també implementat:

```
/** Torna true si n es troba en algun node de la llista,
 *  false en cas contrari. No modifica el pI. */
public boolean conte(int n)
```

**Es demana:** implementar un mètode estàtic (en una classe distinta de `ListPIIntLinked`) tal que:

- Reba com arguments dos objectes de la classe `ListPIIntLinked`, anomenats `a` i `b`.
- Ha d'inserir en la llista `a` només les dades emmagatzemades en la llista `b` que no es troben prèviament emmagatzemades en la llista `a`.
- La inserció en la llista `a` es farà davant de l'element assenyalat pel `pI`, mantenint-se la posició del `pI`.
- En la llista `b` es pot modificar la posició del seu `pI`.
- En la implementació, s'ha d'usar el mètode `conte`.

**Solució:**

```
public static void inserir_nous(ListPIIntLinked a, ListPIIntLinked b) {
    b.begin();
    while (!b.isEnd()) {
        int i = b.get();
        if (!a.conte(i)) { a.insert(i); }
        b.next();
    }
}
```

**P2 - Curs 14/15:** 2.5 punts

Donada una `StackIntLinked p` i un enter `x`, **es demana:** escriure un mètode estàtic (en una classe distinta de `StackIntLinked`), tal que:

- Calcule i torne el número d'aparicions de `x` en `p`.
- Ha de deixar la pila `p` en l'estat en què estava inicialment.

**Solució:**

```
public static int numAparicionsEnPila(StackIntLinked p, int x) {
    int n = 0;
    if (!p.empty()) {
        int aux = p.pop();
        n = numAparicionsEnPila(p, x);
        if (aux == x) { n++; }
        p.push(aux);
    }
    return n;
}
```

Alternativament, usant un array auxiliar:

```
public static int numAparicionsEnPila(StackIntLinked p, int x) {
    int[] aux = new int[p.size()];
    int n = 0, i = 0;
    while (!p.empty()) {
        aux[i] = p.pop();
        if (aux[i] == x) { n++; }
        i++;
    }
    for (i = aux.length - 1; i >= 0; i--) { p.push(aux[i]); }
    return n;
}
```

Alternativament, usant una pila auxiliar:

```
public static int numAparicionsEnPila(StackIntLinked p, int x) {
    StackIntLinked aux = new StackIntLinked();
    int n = 0;
    while (!p.empty()) {
        int e = p.pop();
        if (e == x) { n++; }
        aux.push(e);
    }
    while (!aux.empty()) { p.push(aux.pop()); }
    return n;
}
```

**RecP2 - Curs 14/15:** 3 punts

**Es demana:** En la classe `ListPIIntLinked`, implementar un mètode d'instància amb perfil:

```
public void esborrarEnrere()
```

que, amb un cost com a molt lineal, permeta l'esborrament cap enrere, açò és, que esborre l'element anterior al del punt d'interès.

En cas que la llista estiga buida o que, en general, el punt d'interès es trobe a l'inici, haurà de llançar l'excepció `NoSuchElementException` amb el missstge `"PI a l'inici"`.

En cas contrari, després de l'esborrament, el punt d'interès (`pI`) continua inalterat sobre el mateix element, i l'anterior al punt d'interès (`prevPI`) retrocedeix una posició.

Exemple: Si la llista és `2 3 1 8 9` (el `pI` està sobre el 8), després de l'esborrament ha de quedar `2 3 8 9`.

**Nota:** Només es permet accedir als atributs de la classe, quedant totalment prohibit l'accès als seus mètodes.

**Solució:**

```
/** Elimina l'element anterior al punt d'interès. Si la llista està buida o el
 *  punt d'interès està a l'inici, llança NoSuchElementException. */
public void esborrarEnrere() {
    if (prevPI == null) { throw new NoSuchElementException("PI a l'inici"); }
    if (prevPI == first) {
        first = prevPI.next;
        prevPI = null;
    }
    else {
        NodeInt aux = first;
        while (aux.next != prevPI) { aux = aux.next; }
        aux.next = prevPI.next;
        prevPI = aux;
    }
    size--;
}
```

Noteu que el cost depèn, a més de la size $n$ de la llista, de la posició del `pI`. En el cas pitjor, quan el `pI` està al final de la llista, $T^p(n) \in \Theta(n)$.

# Curs 2013/14

**P2 - Curs 13/14:** 2 punts

A una classe `ExamenEstructures` es desitja afegir un mètode `incrementaParells` estàtic que donada una cua d'enters `q` implementada mitjançant una seqüència de nodes enllaçats, torne una nova cua d'enters on s'hauran canviat els números parells sumant-los 1, deixant els números imparells sense tocar. De manera que si la cua original és:

```
<- 5 10 14 13 22 31 <-
```

la cua a tornar ha de contindre els següents números (en qualsevol ordre):

```
<- 5 11 15 13 23 31 <-
```

**Nota:** Per resoldre aquest problema no es podrà gastar cap estructura de dades addicional.

a) (1.75 punts) Implementa el mètode `incrementaParells`.

b) (0.25 punts) Modifica el mètode anterior en cas que la cua s'implemente mitjançant un array.

**Solució:**

a) Una solució possible (iterativa) entre altres pot ser:

```
public static QueueIntLinked incrementaParells(QueueIntLinked q) {
    QueueIntLinked q1 = new QueueIntLinked();
    int n = q.size();
    for (int i = 0; i < n; i++) {
        int x = q.remove();
        if (x % 2 == 0) { q1.add(x + 1); }
        else { q1.add(x); }
        q.add(x);
    }
    return q1;
}
```

b) Bastaria amb canviar en la capçalera i declaracions de variables el nom de la classe `QueueIntLinked` per `QueueIntArray` i usar també la constructora de `QueueIntArray`.

**P2 - Curs 13/14:** 3 punts

Es desitja modificar el comportament de l'operació `add(int)` en la classe `QueueIntLinked` de forma que els elements es mantinguen de forma ordenada ascendentment; això és, l'element **primer** de la cua serà el de menor valor d'entre tots, mentre que l'**últim** serà el de valor major.

**Es demana** construir el mètode `add(int)` tenint en compte la definició anterior.

**Solució:**

```
public void add(int val) {
    NodeInt ant = null, aux = first;
    while (aux != null && val > aux.data) { ant = aux; aux = aux.next; }

    NodeInt nou = new NodeInt(val, aux);
    if (ant == null) { first = nou; }
    else { ant.next = nou; }

    if (aux == null) { last = nou; }
    size++;
}
```

**P2 - Curs 13/14:** 2 punts

Es vol implementar un mètode anomenat `comptar` tal que:

- Ha de rebre com arguments una llista `l` pertanyent a la classe `ListPIIntLinked`, i dos enters `i`, `j`, que es considera que delimiten un rang de valors `[i,j]`.
- Ha de retornar el nombre d'elements en `l` que estiguen dins del rang `[i,j]`.

S'ha de suposar que el mètode s'està escrivint dins d'una classe diferent a `ListPIIntLinked`.

**Solució:**

```
public static int comptar(ListPIIntLinked l, int i, int j) {
    int compt = 0;
    l.begin();
    while (!l.isEnd()) {
        int x = l.get();
        if (x >= i && x <= j) { compt++; }
        l.next();
    }
    return compt;
}
```

**RecP2 - Curs 13/14:** 2.5 punts

Considerar la següent classe `NodePersona`:

```
class NodePersona {
    int dni;
    String nom;
    NodePersona next;

    NodePersona(int i, String s, NodePersona n) {
        dni = i; nom = s; next = n;
    }
}
```

**Es demana** implementar els següents mètodes estàtics en la classe `NodePersona`:

1. (1.25 punts) Un mètode anomenat `comptar` tal que, donada una seqüència de `NodePersona` i un `String`, torne el número de nodes en la seqüència tals que l'atribut `nom` del node continga l'`String`. En la classe `String` hi ha un mètode, amb perfil `boolean contains(String s)`, que comprova si l'`String` que l'invoca conté a `s`.
2. (1.25 punts) Un mètode anomenat `cercar` tal que, donada una seqüència de `NodePersona` i un `int`, busque si en la seqüència existeix algun node amb atribut `dni` igual a aquest `int`. En cas de trobar el node, torna el valor del seu atribut `nom`; en cas de no trobar-lo, torna la cadena `"Persona desconeguda"`.

**Solució:**

```
public static int comptar(NodePersona n, String s) {
    int compt = 0;
    NodePersona aux = n;
    while (aux != null) {
        if (aux.nom.contains(s)) { compt++; }
        aux = aux.next;
    }
    return compt;
}
```

```
public static String cercar(NodePersona n, int i) {
    NodePersona aux = n;
    while (aux != null && aux.dni != i) {
        aux = aux.next;
    }
    if (aux != null) { return aux.nom; }
    else { return "Persona desconeguda"; }
}
```

**RecP2 - Curs 13/14:** 2.5 punts

Es desitja afegir a la classe `ListPIIntLinked` un nou mètode d'inserció en el punt d'interès (a més del mètode `insert(int)` ja definit en la mateixa) anomenat `inserirSenseRepetir`, el qual realitze la inserció d'un element només si no es troba en la llista; en cas contrari ha de llançar una excepció `IllegalArgumentException` (predefinida de Java i derivada de `RuntimeException`), amb el missatge `"Ja està l'element "` seguit de l'enter que s'ha tractat d'inserir. Per exemple:

- Si la llista és `9 1 4 2 9` i es vol inserir el nou element `0`, la llista ha de quedar `9 1 0 4 2 9`.
- Si la llista és `9 1 4 2 9` i es vol inserir el nou element `2`, la llista ha de quedar `9 1 4 2 9` i s'ha de llançar l'excepció amb el missatge: `Ja està l'element 2`.

**Es demana:**

1. (2.25 punts) Escriure el mètode `inserirSenseRepetir` segons la descripció anterior. Notar que si es verifica que `e` no existeix prèviament en la llista en ninguna posició, es pot usar el mètode `inserir` de la classe.

   **Solució:** Solució d'entre les moltes possibles:

   ```
   public void inserirSenseRepetir(int e) {
       NodeInt aux = first;
       while (aux != null && aux.data != e) { aux = aux.next; }
       if (aux != null) {
           throw new IllegalArgumentException("Ja està l'element " + e);
       }
       this.insert(e);
   }
   ```

2. (0.25 punts) Si aquest mètode s'invocara en un mètode `main`, seria obligatori escriure la crida al mètode dins d'una instrucció `try/catch`, o en el seu defecte modificar la capçalera de `main`? Raonar la resposta.

   **Solució:** No, perquè no llança cap excepció *checked*.

**RecP2 - Curs 13/14:** 2.5 punts

Donada una classe `Examen`, es desitja escriure en ella un mètode per a obtenir el valor major d'una `QueueIntLinked` `c`. En acabar l'operació, `c` ha de ser igual que ho era inicialment. Si la cua està buida ha de llançar l'excepció `NoSuchElementException` amb el missatge `"Cua buida:  màxim no definit"`. Per exemple:

- Si `c` és `<- 4 -2 9 8 <-`, ha de tornar `9` i deixar `c` com `<- 4 -2 9 8 <-`.
- Si `c` és `<- -2 <-`, ha de tornar `-2` i deixar `c` com `<- -2 <-`.
- Si `c` és `<- <-`, `c` ha de seguir sent buida, i es llança l'excepció.

**Solució:**

```
public static int maxim(QueueIntLinked c) {
    int n = c.size();
    if (n == 0) { throw new NoSuchElementException("Cua buida: màxim no definit"); }
    int e = c.remove();
    int maxim = e;
    c.add(e); n--;
    while (n > 0) {
        e = c.remove();
        if (e > maxim) { maxim = e; }
        c.add(e); n--;
    }
    return maxim;
}
```

# Curs 2012/13

**P2 - Curs 12/13:** 2.5 punts

Donada una seqüència enllaçada d'enters `seq` i un enter `x`, **es demana** un mètode amb el següent perfil:

```
public static int ultimaAparicioDe(NodeInt seq, int x)
```

que torne la posició de l'última aparició de `x` en la seqüència i -1 si no està, entenent 0 com la primera posició.

**Solució:**

```
public static int ultimaAparicioDe(NodeInt seq, int x) {
    NodeInt actual = seq;
    int comptador = 0, ultimaAparicio = -1;
    while (actual != null) {
        if (actual.data == x) { ultimaAparicio = comptador; }
        comptador++;
        actual = actual.next;
    }
    return ultimaAparicio;
}
```

**P2 - Curs 12/13:** 2.5 punts

Donada una pila d'enters no buida, **es demana** un mètode **recursiu** amb el següent perfil:

```
public static void esborraBase(StackIntLinked p)
```

que la modifique eliminant l'element de la seua base (l'element més antic de la pila).

**Solució:**

```
/** p és una Pila no buida */
public static void esborraBase(StackIntLinked p) {
    if (p.size() > 1) {
        int x = p.pop();
        esborraBase(p);
        p.push(x);
    }
    else { p.pop(); }
}
```

**P2 - Curs 12/13:** 3 punts

Donades dues llistes amb punt d'interès d'enters, `llista1` i `llista2`, ambdues amb els seus elements en ordre estrictament creixent, **es demana** un mètode amb el següent perfil:

```
/** llista1, llista2 estan en ordre estrictament creixent */
public static ListPIIntLinked unio(ListPIIntLinked llista1, ListPIIntLinked llista2)
```

que calcule la unió de les dues llistes. La llista resultant també haurà de quedar en ordre estrictament creixent.

**Exemple:**
Siga una `llista1` amb els valors: `1, 3, 5, 7, 9, 11, 13, 15, 17, 19`
Siga una `llista2` amb els valors: `1, 4, 7, 10, 13, 16, 19, 21, 27`
El resultat de `unio(llista1,llista2)` ha de ser una llista amb els valors:
`1, 3, 4, 5, 7, 9, 10, 11, 13, 15, 16, 17, 19, 21, 27`

**Solució:**

```
/** llista1, llista2 estan en ordre estrictament creixent */
public static ListPIIntLinked unio(ListPIIntLinked llista1, ListPIIntLinked llista2) {
    ListPIIntLinked li = new ListPIIntLinked();
    llista1.begin();  llista2.begin();
    while (!llista1.isEnd() && !llista2.isEnd()) {
        int i = llista1.get(), j = llista2.get();
        if (i < j) { li.insert(i); llista1.next(); }
        else if (i > j) { li.insert(j); llista2.next(); }
        else { li.insert(i); llista1.next(); llista2.next(); }
    }

    while (!llista1.isEnd()) {
        li.insert(llista1.get());
        llista1.next();
    }

    while (!llista2.isEnd()) {
        li.insert(llista2.get());
        llista2.next();
    }
    return li;
}
```

**RecP2 - Curs 12/13:** 2.5 punts

Siga la classe `ListPIIntLinkedOrd` una classe molt pareguda a la classe `ListPIIntLinked` presentada en el tema 5. En les llistes de la classe `ListPIIntLinkedOrd` els valors enters estan en ordre estrictament creixent. El constructor i tots els mètodes d'aquesta classe (excepte `insert`) tenen la mateixa funcionalitat que en la classe `ListPIIntLinked`.

**Es demana:** implementar, **sense fer ús dels mètodes de la classe (usant només referències)**, el mètode:

```
public void insert(int x)
```

per tal que inserisca `x` ordenadament en la llista. El punt d'interès es situarà a la dreta de l'element que s'insereix.

**Solució:**

```
public void insert(int x) {
    this.pI = this.first; this.prevPI = null;
    while (this.pI != null && this.pI.data < x) {
        this.prevPI = this.pI;
        this.pI = this.pI.next;
    }
    if (this.pI == null || this.pI.data > x) {
        NodeInt nou = new NodeInt(x, this.pI);
        if (this.pI == this.first) { this.first = nou; }
        else { this.prevPI.next = nou; }
        this.prevPI = nou;
        this.size++;
    }
}
```

**RecP2 - Curs 12/13:** 2.5 punts

Donada una llista amb punt d'interès `ListPIIntLinked l` i un enter `x`, **es demana** implementar un mètode estàtic `esborrar` que torne una nova `ListPIIntLinked` amb els elements de la llista `l` iguals a `x`. A més, haurà d'eliminar aquests valors de la llista `l`.

Per exemple, si la llista `l` conté els valors 2, 3, 4, 3, 7, 5 i 3, després de l'execució de la crida `ListPIIntLinked lis = esborrar(l,3)`, la llista `lis` conté els valors 3, 3 i 3 i la llista `l` conté els valors 2, 4, 7 i 5.

**S'usaran únicament els mètodes públics de la classe `ListPIIntLinked`.**

**Solució:**

```
public static ListPIIntLinked esborrar(ListPIIntLinked l, int x) {
    ListPIIntLinked llista = new ListPIIntLinked();
    l.begin();
    while (!l.isEnd()) {
        if (l.get() == x) {
            llista.insert(x);
            l.remove();
        }
        else { l.next(); }
    }
    return llista;
}
```

**RecP2 - Curs 12/13:** 2.5 punts

Donada una `StackIntLinked p` i un enter `x`, **es demana** implementar un mètode estàtic **recursiu** amb el següent perfil:

```
public static void eliminarMenorsQue(StackIntLinked p, int x)
```

que elimine de `p` els elements menors que `x`. Per exemple, si la pila `p` conté els valors: 3, 6, 7, 2, 5 i 4, després de l'execució de la crida `eliminarMenorsQue(p,5)`, la pila conté els valors 6, 7 i 5, tal i com es mostra a continuació:

```
p abans de l'execució                    p després de l'execució
        3
        6
        7
        2        ==== eliminarMenorsQue(p,5) ====>        6
        5                                                 7
        4                                                 5
       ---                                               ---
```

**S'usaran únicament els mètodes públics de la classe `StackIntLinked`.**

**Solució:**

```
private static void eliminarMenorsQue(StackIntLinked p, int x) {
    if (!p.empty()) {
        int aux = p.pop();
        eliminarMenorsQue(p,x);
        if (aux >= x) { p.push(aux); }
    }
}
```

# Curs 2011/12

**P2 - Curs 11/12:** 3.0 punts

Es desitja afegir a la classe `ListPIIntLinked` un parell de nous mètodes, **interns**, que permitisquen obtindre una llista a partir dels elements d'un array i viceversa. **Aquestes noves operacions hauran de fer-se sense fer servir les operacions públiques ja existents en la classe**.

Mitjançant aquests mètodes es podrà ordenar els elements d'una Llista amb punt d'interès transformant-la en un array, ordenant-lo i, fet això, obtenint una nova Llista amb punt d'interès usant l'array. Per a això, **es demana** resoldre els següents apartats:

1. Mètode `toArray()`, que haurà de tornar un array que continga exactament, i en el mateix ordre original, els elements de la `ListPIIntLinked`.
2. Constructor `ListPIIntLinked(int[] a)`, que donat un array d'enters construïsca una `ListPIIntLinked` amb els elements que es troben en l'array, respectant el seu orde.
3. Suposeu la classe `ListPIIntLinked` completada amb els dos mètodes anteriors. Suposeu, a més, que en una classe `ProvaLlista` es disposa ja fet d'un mètode d'ordenació ràpida d'arrays amb el perfil següent:

   ```
   public static void ordFussio(int[] a, int ini, int fi)
   ```

   Escriu les instruccions necessàries per a ordenar, en aquesta classe `ProvaLlista`, certa `ListPIIntLinked` emmagatzemada en una variable anomenada `laLlista`.

**Solució:**

```
public int[] toArray() {
    int[] aux = new int[size];
    int k; NodeInt p;
    for (p = first, k = 0; p != null; p = p.next, k++) { aux[k] = p.data; }
    return aux;
}

public ListPIIntLinked(int[] a) {
    this.first = null;
    for (int k = a.length - 1; k >= 0; k--) { first = new NodeInt(a[k], first); }
    this.size = a.length;
    this.pI = first; this.prevPI = null;     // pI al començament
}

// seqüència d'instruccions per a ordenar laLlista:
int[] copia = laLlista.toArray();
ordFussio(copia, 0, copia.length - 1);
laLlista = new ListPIIntLinked(copia);
```

**P2 - Curs 11/12:** 1.5 punts

Tenint en compte la definició de `StackIntLinked` vista en classe i accedint a la seva representació interna, **es demana** escriure un mètode:

```
public void cimBase()
```

que intercanviï el valor de l'element al cim de la pila amb el de la base de la mateixa (el més antic). Precondició de la operació serà que la pila no estiga buida.

**Solució:**

```
/** Precondició: !empty() */
public void cimBase() {
    NodeInt aux = top;
    int e = aux.data;
    while (aux.next != null) { aux = aux.next; }
    top.data = aux.data;
    aux.data = e;
}
```

**P2 - Curs 11/12:** 2.5 punts

Donades dues Llistes amb punt d'interès `ListPIIntLinked` `la` i `lb`, que es troben ordenades ascendentment i que no contenen elements duplicats, **es demana** fer un mètode anomenat `interseccio` que torne una nova `ListPIIntLinked` que continga tan sols els elements que es troben tant en `la` com en `lb`. Aquesta nova llista tampoc haurà de contindre elements duplicats.

Si no existeixen elements en comú la nova `ListPIIntLinked` estarà buida.

**Exemple:** Si inicialment es té que:

```
la = 2 4 6 7 9 11 33 45 67 112 129 310 516 555 610
lb = 8 11 22 33 44 45 46 112 113
```

Com a efecte de l'execució del mètode que es demana, la llista resultant serà: `11 33 45 112`

**Solució:**

```
/** Precondició: la i lb ordenades ascendentment, sense repetits */
public static ListPIIntLinked interseccio(ListPIIntLinked la, ListPIIntLinked lb) {
    ListPIIntLinked lc = new ListPIIntLinked();
    la.begin(); lb.begin();
    while (!la.isEnd() && !lb.isEnd()) {
        int a = la.get(), b = lb.get();
        if (a < b) { la.next(); }
        else if (b < a) { lb.next(); }
        else { lc.insert(a); la.next(); lb.next(); }
    }
    return lc;
}
```

**RecP2 - Curs 11/12:** 1.5 punts

Considerant la implementació de les classes `NodeInt` i `StackIntLinked` explicades en classe, què mostra per pantalla el següent programa?

```
public class Piles {
    public static void main(String[] args) {
        StackIntLinked p1 = new StackIntLinked();
        for (int i = 1; i <= 10; i++) { p1.push(i); }
        StackIntLinked p2 = new StackIntLinked();
        while (!p1.empty()) {
            int valor = p1.pop();
            if (valor % 2 == 0) { p2.push(valor); }
            else { System.out.print(" " + valor); }
        }
        while (!p2.empty()) { System.out.print(" " + p2.pop()); }
    }
}
```

**Solució:** 9 7 5 3 1 2 4 6 8 10

**RecP2 - Curs 11/12:** 3 punts

Per tal de representar paraules com seqüències enllaçades de valors de tipus `char`, es suposa ja implementada la classe `NodeChar` (anàloga a la classe `NodeInt` vista en classe), amb atributs `dada` de tipus `char` i `seguent` de tipus `NodeChar`, i amb les dues operacions constructores habituals definides en aquest tipus de classe. **Es demana** escriure un mètode estàtic `corregir`, en una classe inclosa en el mateix paquet que la classe `NodeChar`, amb el següent perfil:

```
public static NodeChar corregir(NodeChar p)
```

que, donada una paraula `p` (de tipus `NodeChar`, amb al menys 1 caràcter), la corregisca substituint totes les ocurrències del parell de caràcters consecutius **'n' 'y'** pel caràcter **'ñ'**. Per exemple, les paraules "cucanya" i "nyonyería", serien "cucaña" i "ñoñería".

**Solució:**

```
/** la paraula té al menys 1 caràcter */
public static NodeChar corregir(NodeChar p) {
    NodeChar aux = p.next, ant = p;
    while (aux != null) {
        if (ant.data == 'n' && aux.data = 'y') {
            ant.data = 'ñ';
            ant.next = aux.next;
            aux = ant.next;
        }
        else {
            ant = aux;
            aux = aux.next;
        }
    }
    return p;
}
```

**RecP2 - Curs 11/12:** 2.5 punts

Donada una llista amb punt d'interès `ListPIIntLinked l`, **es demana** escriure un mètode estàtic `eliminarNeg` que elimine els valors negatius d'aquesta llista, fent ús exclusivament de les operacions públiques definides en la classe `ListPIIntLinked`, sense accedir a la seua representació interna.

**Exemple:** Si inicialment es té que `l = 3 -2 5 -7 -8 1 -10`, com a efecte de l'execució del mètode que es demana, la llista resultant serà `l = 3 5 1`.

**Solució:**

```
public static void eliminarNeg(ListPIIntLinked l) {
    l.begin();
    while (!l.isEnd()) {
        if (l.get() < 0) { l.remove(); }
        else { l.next(); }
    }
}
```

## Curs 2010/11

La classe `ListIntLinked` té dos atributs d'instància privats: `first` de tipus `NodeInt` i `size` de tipus `int`, i implementa les següents operacions públiques:

- `ListIntLinked`, constructora.
- `size()`, torna la talla.
- `empty()`, diu si està buida.
- `search(int x)`, torna la posició que ocupa un element `x` donat. Torna -1 si no es troba.
- `get(int i)`, torna l'element que està en una posició `i` donada, $0 \leq$ `i` $<$ `n` sent `n` la talla de la llista.
- `insert(int i, int x)`, insereix un nou element `x` en una posició `i` donada, $0 \leq$ `i` $\leq$ `n` sent `n` la talla de la llista; quan `i == n`, s'insereix al final.
- `remove(int i)`, torna i elimina l'element que ocupa una posició `i` donada, $0 \leq$ `i` $<$ `n` sent `n` la talla de la llista.

**P2 - Curs 10/11:** 3 punts

Prenent en consideració la definició de `ListIntLinked` vista en classe,

**Es demana**: fer un mètode:

```
public void inserirSenseRepetits(int val, boolean davant)
```

per a inserir un element `val` en la `ListIntLinked` en el cas de que no existira prèviament en ella, bé al començament de la llista, com a primer element, si l'argument `davant` és `true`, bé per la part de darrere, a la fi de la llista, cas de que l'argument `davant` siga `false`. La `ListIntLinked` romandrà sense canvi en el cas de que l'element `val` ja existira inicialment en ella. **Nota:** aquesta operació haurà de tindre un **cost lineal**.

**Solució:**

```
/** Insereix un nou element sols si no existeix(sense duplicats).
 *  Be al comencament (davant es true), be a la fi (davant es false). */
public void inserirSenseRepetits(int val, boolean davant) {
    NodeInt p = first, q = null;
    while (p != null && p.data != val) {
        q = p; p = p.next;
    }
    if (p == null) {
        size++;
        if (q == null) { first = new NodeInt(val); }
        else if (davant) { first = new NodeInt(val, first); }
        else { q.next = new NodeInt(val); }
    }
}
```

**P2 - Curs 10/11:** 2 punts

Suposant ja fetes les classes:

- `NodeStr`, amb atributs `data` de tipus `String` i `next` de tipus `NodeStr`, i amb les dues operacions constructores habituals definides en aquests tipus de classes.
- `ListStrLinked`, mitjançant la qual es manté una llista enllaçada d'elements de tipus `NodeStr` i amb totes les operacions definides en classe per a les llistes.

**Es demana:** Fer una operació, dintre de la classe `ListStrLinked`, que determine si una llista d'eixe tipus està o no ordenada ascendentment. Una llista de paraules tal com una `ListStrLinked`, està ordenada ascendentment si per a qualsevol parella d'elements consecutius de la llista, el primer element és anterior o igual al segon. A més, per definició, una llista buida o amb tan sols un element, està ordenada. **Nota:** aquesta operació haurà de tindre un **cost lineal**.

**Solució:**

```
/** True sii la llista està ordenada ascendentment */
public boolean estaOrdenat() {
    if (size == 0 || size == 1) { return true; }
    else {
        NodeStr p = first.next;
        NodeStr q = first;
        while (p != null && q.data.compareTo(p.data) <= 0) {
            q = p; p = p.next;
        }
        return p == null;
    }
}
```

**P2 - Curs 10/11:** 3 punts

Suposant ja definida la classe `NodeInt`, vista en classe, escriure una classe `ExamenIntEnla` amb les operacions següents:

- `ExamenIntEnla()`, constructora de l'estructura buida.
- `inserirAlPrincipi(int)`, insereix al començament de l'estructura el valor que es rep com a argument.
- `inserirAlFi(int)`, insereix al final de l'estructura el valor que es rep com a argument.

**Es demana:** Implementar la classe `ExamenIntEnla`, declarant per a ella el(s) atribut(s) necessari(s), i implementant totes les operacions demanades. **Nota:** totes les operacions hauran de tindre un **cost constant**.

**Solució:**

```
package linear;
/**
 * @author (PRG - ETSINF - DSIC)
 * @version (examen juny 2011)
 */
public class ExamenIntEnla {
    private NodeInt ini;
    private NodeInt ult;

    public ExamenIntEnla() { ini = ult = null; }

    public void inserirAlPrincipi(int x) {
        if (ini == null) { ult = ini = new NodeInt(x); }
        else { ini = new NodeInt(x, ini); }
    }

    public void inserirAlFi(int x) {
        if (ini == null) { ult = ini = new NodeInt(x); }
        else { ult = ult.next = new NodeInt(x); }
    }
}
```

**P2 - Curs 10/11:** 2 punts

Partint de la classe `ListIntLinked`, i fent-la servir exclusivament, es desitja fer una nova classe `StackIntLinked2` que emprant les operacions de `ListIntLinked` implemente totes les operacions d'una Pila, això és, les següents:

- `StackIntLinked2()`, constructora.
- `size()`, torna la talla.
- `empty()`, diu si està buida.
- `peek()`, torna, si existeix, el cim.

- `push(int)`, empila l'element que se li dona.
- `pop()`, desempila i torna, si existeix, l'element al cim.

**Es demana:** Implementar, fent servir `ListIntLinked`, la classe `StackIntLinked2`, declarant per a ella **un únic atribut** de tipus `ListIntLinked`, i implementant totes les operacions demanades. **Nota:** Totes les operacions hauran de tindre un **cost constant**.

**Solució:**

```
package linear;
/**
 * StackIntLinked2 implementa una PilaInt mitjancant una ListIntLinked
 * @author (PRG - ETSINF - DSIC)
 * @version (examen juny 2011)
 */
public class StackIntLinked2 {
    private ListIntLinked laLlista;

    public StackIntLinked2() { laLlista = new ListIntLinked(); }

    public int size() { return laLlista.size(); }

    public boolean empty() { return laLlista.empty(); }

    public int peek() { return laLlista.get(0); }

    public void push(int x) { laLlista.insert(0, x); }

    public int pop() { return laLlista.remove(0); }
}
```