

Cognoms

Nom

DNI

Observació: Heu de fer servir els espais indicats per entrar la resposta d'alguns apartats dels problemes. Noteu que podeu deixar en blanc alguna de les caixetes si creieu que no cal cap instrucció en aquell punt.

Problema 1 (5 punts)

Una *escala* d'una llista v de naturals és una subllista d'elements consecutius de v on cada element és menor o igual que el seu següent. Construïu un programa que llegeixi una seqüència de naturals acabada en 0, l'emmagatzemi en una llista d'enters, calculi la longitud de l'escala de longitud màxima d'aquesta llista i la posició del primer element d'aquesta escala, i escrigui l'escala de longitud màxima.

Per facilitar l'especificació i la implementació de les operacions descrites en aquest problema hem definit els tipus `Li`, `It` i `CI`.

```
typedef list<int> Li;
typedef list<int>::iterator It;
typedef list<int>::const_iterator CI;
```

El vostre programa ha d'utilitzar la funció `max_long_esc` especificada a continuació.

```
pair<int, CI> max_long_esc(const Li& v);
/* Pre: Cert. */
/* Post: El primer component del resultat és la longitud de l'escala de v de
longitud màxima. El segon component del resultat apunta al primer element de
l'escala de longitud màxima de v, i en cas d'empat apunta al primer element de
l'escala de longitud màxima de v més propera a v.begin(). Si v és buida, el
primer component del resultat és 0 i el segon component apunta a v.end(). */
```

Per exemple, si la seqüència d'entrada és 1 2 1 2 3 1 2 2 4 0 el programa ha de llegir la seqüència, emmagatzemar la llista (1,2,1,2,3,1,2,2,4) en una variable v de tipus `list<int>`, cridar la funció `max_long_esc` amb v , i escriure la subllista 1 2 2 4

Semblantment, si la seqüència d'entrada és 4 1 2 3 2 3 1 3 4 0 el programa ha de llegir la seqüència, emmagatzemar la llista (4,1,2,3,2,3,1,3,4) en una variable v de tipus `list<int>`, cridar la funció `max_long_esc` amb v , i escriure la subllista 1 2 3

Finalment, si la seqüència d'entrada és 4 3 2 1 0 el programa ha de llegir la seqüència, emmagatzemar la llista (4,3,2,1) en una variable v de tipus `list<int>`, cridar la funció `max_long_esc` amb v , i escriure la subllista 4

Concretament es demana

Problema 1.1 (2.5 punts): Implementar iterativament la funció `max_long_esc`. Es valorarà l'eficiència de la vostra implementació.

Problema 1.2 (1.5 punts): Justificar la correctesa de la vostra implementació de la funció `max_long_esc`. Heu de donar l'invariant, la justificació del cos del bucle i de les inicialitzacions, i la funció de fita.

Problema 1.3 (0.25 punts): Completar l'especificació i la implementació de la següent acció.

```
void llegir (  ) {  
    /* Pre: Hi ha una seqüència de naturals acabada en 0 al canal d'entrada estàndard. */  
    /* Post: Els elements de la seqüència s'han emmagatzemat a la llista v. */  
      
    int x;  
    cin >> x;  
    while (x ≠ 0) {  
          
        cin >> x;  
    }  
}
```

Problema 1.4 (0.5 punts): Especificar e implementar una acció anomenada *escriure* que escrigui pel canal estàndard de sortida una subllista d'una llista donada coneixent la posició del primer element de la subllista (un iterador de tipus `CIt`) i la longitud de la subllista. Heu de donar la capçalera, precondition, postcondició i implementació de l'acció *escriure*.

```
void escriure (  ) {  
    /* Pre:  
     */  
    /* Post:  
     */  
      
}
```

Problema 1.5 (0.25 punts): Completar el programa principal cridant a les operacions `max_long_esc`, `llegir` i `escriure`.

```
int main() {  
    list <int> v;  
    llegir (v);  
      
    escriure (  );  
}
```

Cognoms

Nom

DNI

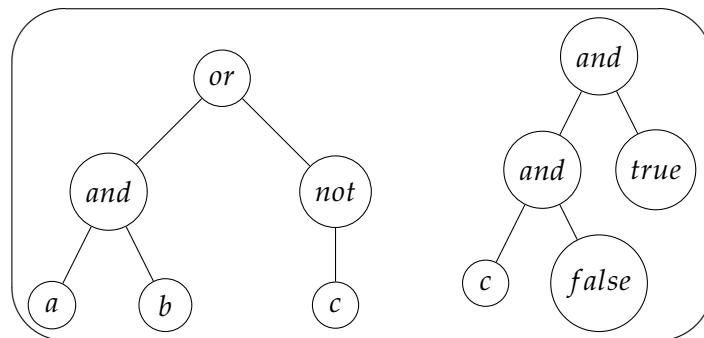
Problema 2 (5 punts)

En un llenguatge de programació com ara C++ podem definir expressions booleans com les següents

- (a and b) or not c
- (c and false) and true

Si a i b són certs i c és fals, el resultat d'avaluar la primera expressió és cert, mentre que el resultat d'avaluar la segona expressió és fals.

Tota expressió booleana com les anteriors es pot representar mitjançant un arbre de cadenes de caràcters (Arbre<string>) on els nodes són operadors booleans (and, or, not), identificadors de variables booleans, o constants booleans (true, false). Per exemple, les expressions anteriors donen lloc als arbres



Suposem que les expressions booleans estan ben formades, és a dir, no donarien cap error de compilació en C++. En particular, totes les variables de les expressions estan definides. Per tant, els arbres associats a una expressió booleana són arbres de string no buits amb les següents propietats:

1. Si l'arrel és un operador booleà binari, aleshores els subarbres fill esquerre i fill dret no són buits.
2. Si l'arrel és un operador booleà unari, aleshores el subarbre fill esquerre no és buit però el subarbre fill dret és buit.
3. Les fulles només poden ser constants booleans o variables booleans (no poden ser operadors).
4. Els nodes interns només poden ser operadors booleans.
5. No hi ha cap node de l'arbre que contingui parèntesis.

Problema 2.1 (2.5 punts): Implementeu eficientment la funció avaluar especificada a continuació.

```
bool avaluar(Arbre<string> &a, const Variables& e);
/* Pre: a = A; A és un arbre no buit associat a una expressió booleana ben formada; totes les variables que apareixen a A estan definides al objecte e de tipus Variables i es pot calcular el seu valor cridant el mètode "valor" d'aquest objecte. */
/* Post: Retorna el resultat d'avaluar l'expressió booleana que representa A. */
```

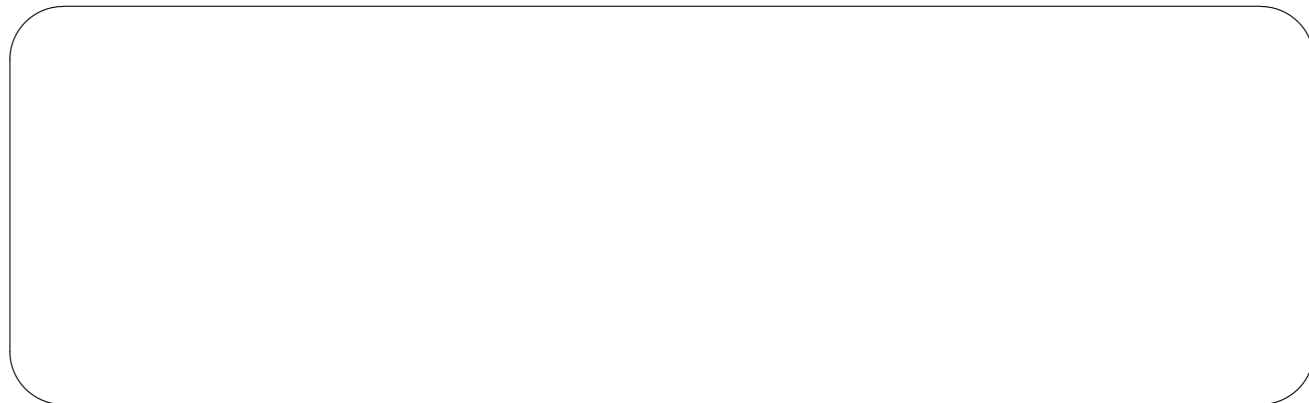
Teniu en compte que C++ avalua les expressions booleans de forma curtcircuitada: si el resultat d'avaluar el primer argument de l'operador **and** és **false**, C++ no avalua el segon argument; i si el resultat d'avaluar el primer argument de l'operador **or** és **true**, C++ no avalua el segon argument. Per exemple, per avaluar l'expressió (a and b) or not c (corresponent a l'arbre esquerre de la figura) C++ no avaluarà el subarbre d'aquest arbre amb arrel **not**. Semblantment, per avaluar l'expressió (c and false) and true (corresponent a l'arbre dret de la figura) C++ no avaluarà els dos subarbres amb arrels **false** i **true**.

Per obtenir el valor d'una variable booleana amb identificador `s` podeu fer servir el mètode públic `valor` de la classe `Variables`.

```
bool valor(const string & s);  
/* Pre: s és l'identificador d'una variable booleana definida al paràmetre implícit. */  
/* Post: El resultat és el valor de la variable amb identificador s al paràmetre implícit. */
```

Observació: No heu d'implementar el mètode `valor` ni la classe `Variables` a la vostra resposta.

Problema 2.2 (0.5 punts): Especifiqueu l'ordre que utilitza el vostre algoritme per tractar els nodes de l'arbre (pre-ordre, in-ordre, post-ordre o per nivells), i perquè.



Problema 2.3 (2 punts): Justifiqueu la correctesa de la vostra implementació de la funció `avaluar`. Escriviu les hipòtesis d'inducció i utilitzeu aquestes hipòtesis d'inducció per justificar la correctesa dels casos recursius (1 punt). Justifiqueu l'acabament de la funció que heu implementat (1 punt).