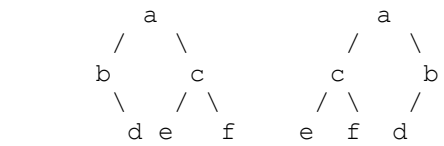
BinTrees isomorfos

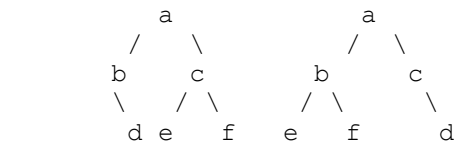
Tiempo estimado: 10 minutos

Diremos que dos BinTree son isomorfos si hay alguna manera de intercambiar arboles izquierdos y derechos de uno de ellos tantas veces como sea necesario de tal manera que los dos árboles sean iguales.

Por ejemplo



son isomorfos, pero



no lo son.

Completa la función dada más abajo para que cumpla su especificación, es decir, indica qué instrucciones deben reemplazar los lugares indicados con números ([---n---]).

```
bool isomorfos(const BinTree<int> & a, const BinTree<int> & b) {
  bool res;
  if (a.empty() or b.empty()) { [--------- 1 -------------] }

 else if (a.value() != b.value()) { [ ------- 2 ----------] }

 else { [ ------- 3 ------- ] };

  return res;
}
```

Cada uno de los lugares señalados (1, 2 y 3) consiste en una o más instrucciones consecutivas y simples del estilo

```
res = ....;
```

Invertir prefijo de lista

Tiempo estimado: 15 min

Completa la función dada más abajo para que cumpla su especificación, es decir, indica qué instrucciones o expresiones deben reemplazar los lugares indicados con números ([---n---]).

```
// Pre: it=IT apunta a un elemento x_i de l o it == l.end()
// y l = L = [x_1,....,x_N] donde N >= 0
void reverse_prefix(list& l, list::iterator it) {
      list::iterator p = [------- 1 --------];
      while ([------ 2 ------]) {
            [---- 3 -----];
      };
};
// Post: l está formada por el prefijo invertido de L
// entre L.begin() e IT, y a continuación, L[IT:);
// esto es, l = [ x_{i-1},...,x_1,x_i,...,x_N]
```

En una solución correcta

[--1--] es una expresión simple
[--2--] es una expresion booleana
[--3--] es una o más instrucciones simples

y el coste del bucle es proporcional al número de elementos que tiene el prefijo de L que se ha de invertir. Además está prohibido añadir expresiones o instrucciones fuera de los lugares indicados (p.e. no se pueden añadir más inicializaciones o instrucciones al terminar el bucle) y se valorará negativamente cualquier solución que modifique *it o *p (por ejemplo, intercambiar *p con *it).

Partición de listas

Tiempo estimado: 15 minutos

Tenemos que diseñar un procedimiento parte_lista que, dados una lista l de enteros y un valor entero x, modifica la lista l para que todos los elementos menores o iguales que x estén delante de todos los elementos mayores que x y nos devuelve un iterador al primer elemento >x, o a l.end() si no hay ningún elemento > x en l. El orden relativo entre los elementos de l no importa y no tiene porque coincidir con el que tuvieran en L.

```
// Pre: l = L
list<int>::iterator parte_lista(list<int>& l, int x);
// Post: l es una permutación de L, it = parte_lista(l,x)==l.end()
// si no hay elementos mayores que x en L o it apunta a un elemento > x,
// todos los elementos en l[it:) son > x, y todos los elementos
// de l[:it) son <= x
```

Tu solución ha de ser iterativa y debe preservar necesariamente este invariante:

1. l es una permutación de L **y**
2. todos los elementos de l[:it1) son menores o iguales que x **y**
3. todos los elementos de l[it2:) son mayores que x, siendo L el valor original de la lista l e it1 e it2 iteradores a elementos de l.

Substitució en un vector

Temps estimat: 15 minuts

Escriu una funció subst_sum que, donat un vector v d'enters v = (v0, ..., v_(n-1)) no buit, substitueix el primer element del vector per la suma de tots els seus elements. La teva solució ha de utilitzar una funció d'immersió i_subst_sum recursiva. Implementa la funció d'immersió, i implementa la función subst_sum utilitzant la funció i_subst_sum, respectant les especificacions donades:

```
// Pre: v = (V_0, ..., V_(N-1)) i N > 0
void subst_sum(vector& v);
// Post: v = (S, V_1, ..., V_(N-1)), ón S = V_0 + ... + V_(N-1)


// Pre: v= (V_0, ..., V_(N-1)), N > 0, 0 <= i < N, s = V_0 + ... + V_(i-1)
void i_subst_sum(vector& v, int i, int& s);
// Post: v = (s,V_1, ..., V_(N-1)), s = V_0 + ... + V_i + ... + V_(N-1)
```

Declarando un método

Tiempo estimado: 2 minutos

Queremos añadir a la clase CjtEstudiants una operación actualiza_notas que dado un vector de N reales actualiza la nota de los primeros N estudiantes de un CjtEstudiants.

```
class CjtEstudiants {
public:
            ...
            actualiza_notas(...)
private:
           ...
};
```

¿Cúal es la cabecera apropiada para actualiza_notas?

- static void actualiza_notas(vector<double>& v);
- void actualiza_notas(vector<double>s& v) const;
- static CjtEstudiants actualiza_notas(vector<double> v);
- CjtEstudiants actualiza_notas(vector<Estudiant>& v) const;
- void actualiza_notas(vector<double> v);
- void actualiza_notas(const vector<double>& v);
- void actualiza_notas(const vector<double> v);
- static CjtEstudiants actualiza_notas(CjtEstudiants& c, vector<double> v);

Funció de fita

Temps estimat: 3 minuts

Donada la següent funció

```
void simdif(const vector<int>& A, const vector<int>& B,

                   vector<int>& C) {
  int i = 0; int j = 0;
  while (i < A.size() and j < B.size()) {
     if (A[i] < B[j]) { C.push_back(A[i]); ++i; }
     else if (A[i] > B[j]) { C.push_back(B[j]); ++j; }
     else { ++i; ++j; }
  }
  ...
}
```

Quina de les següents funcions és una **funció de fita** valida per a demostrar la terminació del bucle principal?

- i+j
- A.size()-i
- A.size()-i > 0 i B.size()-j > 0
- B.size()-j
- C.size()
- A.size()+B.size()-i-j
- min(A.size()-i,B.size()-j)
- max(A.size()-i,B.size()-j)

Imprimendo un árbol

Tiempo estimado: 2 minutos

Si tenemos la función que se muestra a continuación y se le da como entrada el árbol de abajo, ¿qué se imprime en pantalla?

```
void mystery(const BinTree& a) {
      if (not a.empty()) {
          cout << a.value() << endl;
          mystery(a.right());
          mystery(a.left());
 }
}
```

```
          gato

         /    \

        /       \

      leon      mono

      /   \         \
```

```
    /     \          \
  vaca   pato      conejo
           /
          /
       cebra
```

- gato, leon, mono, vaca, pato, conejo, cebra
- gato, mono, leon, conejo, pato, vaca, cebra
- gato, mono, conejo, leon, pato, cebra, vaca
- vaca, leon, cebra, pato, gato, mono, conejo
- cebra, pato, leon, vaca, conejo, mono, gato
- conejo, mono, cebra, pato, vaca, leon, gato
- imprime una permutación de las 7 palabras pero no es ninguna de las otras secuencias
- la función no compila, aunque supongamos que se han hecho todos los #include's necesarios
- la función no se ejecuta correctamente y el programa aborta

Subsecuencia de repetidos

Tiempo estimado: 8 minutos

Tenemos un vector de enteros, y queremos saber la longitud de la subsecuencia más larga de números consecutivos iguales y dónde se inicia dicha subsecuencia. En caso de empate nos interesa la que empiece antes.

Considera el siguiente algoritmo:

```
// Pre: v.size() > 0
int ini=0;
int lmax=1;
int i=1;
int ini_act=0;
while (i < v.size()) {
  if (v[i] != v[i-1]) {
     if (lmax < i-ini_act-1) {
       ini=ini_act; lmax=i-ini_act-1;
     }
  }
  ++i;
}
// Post: la subsecuencia de números iguales más larga en v tiene longitud
// "lmax" y se inicia en la posición "ini" del vector; en caso de empate
// "ini" es la menor posición en la que se inicia una subsecuencia
// de repeticiones de longitud máxima
```

¿Cúal de las siguientes afirmaciones es cierta?

- El algoritmo no es correcto, debería funcionar con vectores vacíos tambien. Solamente será necesario poner lmax = 0; y ini = -1 antes del bucle para arreglarlo.
- El algoritmo no es correcto. Hay casos en los que se cumple la precondición, pero no se cumple la postcondición al terminar el bucle.
- El algoritmo no es correcto, debería empezar inicializando i = 0; y

comparando v[i] con v[i+1] en cada iteración, no v[i] con v[i-1].

- El algoritmo es correcto, tal como puede verificarse usando el invariante

  "la subsecuencia de números iguales más larga en v[0..i] tiene longitud "lmax" y se inicia en la posición "ini" del subvector; en caso de empate "ini" es la menor posición en la que se inicia una subsecuencia de repeticiones de longitud máxima del subvector, 0 <= i <= v.size(), 0 <= ini <= ini_act < v.size()

- El algoritmo no es correcto, hay casos en los que el bucle no termina nunca; pero si el algoritmo termina entonces da la respuesta correcta.

Enviar consulta