Control PRO2 - Torn 1 (Primavera 2017) X27643_ca

Hem decidit estendre la classe `Cjt_estudiants` que heu vist al laboratori amb una nova funcionalitat que assigna automàticament un nombre limitat de beques als estudiants aprovats amb millors notes i, en cas d'empat, als estudiants aprovats amb millors notes en ordre descendent per DNI. Concretament, hem afegit dos mètodes públics a la classe `Cjt_estudiants`: 1) `b_assignades`, que retorna el nombre de beques assignades a estudiants del conjunt (és a dir, el nombre d'estudiants del conjunt que tenen beca); 2) `pos_max_no_becat`, que retorna la posició del millor candidat a obtenir una beca del conjunt que encara no té beca, si existeix algú, o -1 si no hi ha candidats sense beca. Un estudiant és *candidat* a obtenir una beca si està aprovat. Donats dos candidats e_1 i e_2 direm que e_1 és millor que e_2 si e_1 té millor nota que e_2, o si e_1 i e_2 tenen la mateixa nota i el DNI d'e_1 és més gran que el DNI d'e_2. En tot moment el nombre d'estudiants amb beca del conjunt `n_bec` serà més petit o igual al nombre de beques disponibles `MAX_BEC`. A més `n_bec` serà més petit o igual al nombre d'estudiants aprovats `na`. Finalment, si el nombre de estudiants aprovats és major o igual al nombre de beques disponibles `MAX_BEC`, el nombre d'estudiants amb beca del conjunt `n_bec` serà igual a `MAX_BEC`.
Per implementar eficientment aquesta funcionalitat hem modificat la representació i l'invariant de la classe `Estudiant` de la manera descrita a l'arxiu `Estudiant.hh`. En particular, representem la informació sobre si un estudiant té beca o no als objectes de la classe `Estudiant`, i no als objectes de classe `Cjt_estudiants`. Això vol dir que per assignar una beca a l'estudiant situat a la posició `pos` de `vest` hem d'utilitzar la instrucció `vest[pos].modificar_beca(true);` i semblantment per comprovar si té beca o està aprovat. Llegiu amb cura l'arxiu `Estudiant.hh`, especialment les descripcions dels nous atributs, l'invariant i les especificacions de les operacions noves. Les principals novetats de la classe `Estudiant` són:

- hem incorporat un nou atribut `amb_beca` que permet representar informació sobre si l'estudiant paràmetre implícit té beca o no;
- hem afegit el consultor `te_beca` que permet comprovar si l'estudiant paràmetre implícit té beca o no;
- hem afegit el modificador `modificar_beca` que permet modificar la informació sobre si l'estudiant paràmetre implícit té beca o no;
- hem afegit el consultor `aprovat` que permet comprovar si l'estudiant paràmetre implícit està aprovat o no;
- hem afegit el mètode `static` i públic `major_nota_dni` que permet comparar dos estudiants per nota i en cas d'empat per DNI;

També hem modificat la representació i l'invariant de la classe `Cjt_estudiants` de la manera descrita a l'arxiu `Cjt_estudiants.hh`. La principal novetat és que emmagatzemem la posició del millor candidat no becat del conjunt en un nou atribut `i_max_no_becat`, de manera que si en algun moment hi ha una beca disponible coneguem la posició de l'estudiant al qual li hem d'assignar aquesta beca. Llegiu amb cura l'arxiu `Cjt_estudiants.hh`, especialment les descripcions dels nous atributs, l'invariant i les especificacions de les operacions noves. Les principals novetats de la classe `Cjt_estudiants.hh` són:

- hem incorporat un nou atribut `static` i constant `MAX_BEC` que especifica el nombre de beques disponibles;
- hem incorporat un nou atribut `n_bec` que conté el nombre d'estudiants que tenen beca del paràmetre implícit, que equival al nombre de beques assignades a estudiants del p.i. de les `MAX_BEC` beques disponibles;
- hem afegit un consultor `b_assignades` que permet consultar el valor de l'atribut `n_bec`;
- hem incorporat un nou atribut `i_max_no_becat` que conté la posició del millor candidat no becat del paràmetre implícit. Si hi ha candidats no becats al paràmetre implícit, llavors `i_max_no_becat` conté la posició del millor candidat no becat i el seu valor està dins de l'interval $0 \leq$ `i_max_no_becat` $<$ `nest`; en cas contrari, és a dir, si no hi ha candidats sense beca, el valor de l'atribut `i_max_no_becat` és igual a -1;
- hem afegit el consultor públic `pos_max_no_becat`, per consultar la posició del millor candidat no becat. Si hi ha candidats sense beca al conjunt paràmetre implícit retorna `i_max_no_becat`$+1$, i si no hi ha candidats sense beca retorna -1.
- i hem afegit el modificador privat `recalcular_pos_max_no_becat` per recalcular el valor de l'atribut `i_max_no_becat` quan sigui necessari.

Tenint això en compte heu d'implementar eficientment el següent mètode privat sense utilitzar l'operació `sort` de la biblioteca <`algorithm`>:

```
void recalcular_pos_max_no_becat();
/* Pre: cert */
/* Post: Si hi ha candidats sense beca al conjunt paràmetre implícit,
l'atribut i_max_no_becat conté la posició del millor candidat no becat
i el seu valor està dins de l'interval 0 <= i_max_no_becat < nest; si
no hi ha candidats sense beca, l'atribut i_max_no_becat és igual a -1. */
```

i el següent mètode públic sense utilitzar l'operació `sort` de la biblioteca <`algorithm`>. Noteu que quan esborrem un estudiant amb beca del conjunt, la seva beca s'assigna al millor candidat no becat del conjunt, si n'hi ha cap. Si el conjunt no conté candidats no becats, la beca de l'estudiant esborrat no s'assigna a cap estudiant. Obviament, si esborrem un estudiant no becat, el nombre d'estudiants que tenen beca no es modifica.

```
void esborrar_estudiant(int x, bool& trobat);
/* Pre: cert  */
/* Post: Si el paràmetre implícit original contenia un estudiant amb
     DNI = x, trobat és true, el p.i. conté els mateixos estudiants
     que l'original menys l'estudiant amb DNI = x, s'han actualizat
     els estudiants becats del p.i. si ha estat necessari, i s'ha
     actualitzat la posició del millor candidat no becat del p.i. si
     ha estat necessari; en cas contrari, trobat és false i el p.i.
     és igual a l'original. */
```

Observació

Heu de lliurar un fitxer `solucio.cc` amb una implementació eficient de les operacions `recalcular_pos_max_no_becat` i `esborrar_estudiant` que ha de tenir el següent format:

```
#include "Cjt_estudiants.hh"

void Cjt_estudiants::recalcular_pos_max_no_becat() {
  ... // codi de la implementació
}

void Cjt_estudiants::esborrar_estudiant(int x, bool& trobat) {
  ... // codi de la implementació
}
```

Copieu aquesta plantilla en el vostre `solucio.cc` i completeu-la. El vostre `solucio.cc` no pot contenir la implementació d'altres operacions de la classe.
A l'apartat *Public files* del Jutge us proveïm amb material addicional comprimit en un fitxer .tar. Podeu descomprimir aquest fitxer amb la comanda

```
tar -xvf nom_fitxer.tar
```

Aquest material addicional consisteix en els següents fitxers:

- `Cjt_estudiants.hh`: l'especificació Pre/Post de totes les operacions públiques i privades d'aquesta nova versió de la classe `Cjt_estudiants`, així como la definició dels atributs privats. Fixeu-vos que **hem afegit tres atributs** `n_bec, i_max_no_becat` i `MAX_BEC`, i que **hem modificat l'invariant** de la representació de `Cjt_estudiants`. **És molt important que la implementació de les operacions que us hem encarregat tingui en compte i preservi l'invariant de la representació**. Fixeu-vos també que hi ha quatre operacions noves: el modificador privat `recalcular_pos_max_no_becat` i els consultors públics `bec_disp, b_assignades, pos_max_no_becat`.
- `Cjt_estudiants.cc`: la implementació de totes de les operacions de la nova versió de la classe `Cjt_estudiants` tret de les operacions que us demanem.
- `Estudiant.hh`: l'especificació de la classe `Estudiant` i la definició dels seus atributs. Les principals novetats que presenta són un atribut `amb_beca` que indica si s'ha concedit una beca a l'estudiant paràmetre implícit, i els mètode públics `aprovat, te_beca, modificar_beca` i `major_nota_dni`.
- `Estudiant.cc`: la implementació dels mètodes de la classe `Estudiant`.
- `pro2.cc`: un programa principal que podeu fer servir per provar els mètodes públics d'aquesta versió de la classe `Cjt_estudiants`.
- `llegeixme.txt`: instruccions per a generar l'executable del programa `pro2` i provar-lo.

Valorarem positivament que la solució no contingui instruccions (especialment bucles o crides a operacions costoses) ni objectes (especialment vectors o conjunts) innecessaris, que no faci recorreguts quan hauria de fer cerques, i que usi correctament les operacions més eficients de la classe sempre que sigui possible. No es pot emprar cap estructura de dades que no hagi aparegut a les sessions 1-4 de laboratori.

La utilització de l'operació `sort` de la biblioteca `<algorithm>` a l'arxiu `solucio.cc` comportarà una qualificació de 0 a la correcció manual del control.

Quan feu els enviaments el Jutge us indicarà quants jocs de proves passeu i de quin tipus (públic o privat). El joc de proves anomenat `públic` correspon als fitxers `entrada.txt` i `sortida_correcta.txt` de l'apartat *Public files*.

Informació del problema

Autor : Professors de PRO2
Generació : 2017-03-18 21:03:45