Control PRO2 - Turno 1 (Primavera 2017) X27643_es

Hemos decidido extender la clase `Cjt_estudiants` que habéis visto en el laboratorio con una nueva funcionalidad que asigna automáticamente un número limitado de becas a los estudiantes aprobados con mejores notas, y en caso de empate a los estudiantes aprobados con mejores notas en orden descendente de DNI. Concretamente, hemos añadido dos métodos públicos a la clase `Cjt_estudiants`: 1) `b_assignades`, que devuelve el número de becas asignadas a estudiantes del conjunto (es decir, de estudiantes del conjunto que tienen beca); 2) `pos_max_no_becat`, que devuelve la posición del mejor candidato a obtener una beca del conjunto que todavía no tiene beca, si existe alguno, o -1 si no hay candidatos sin beca. Un estudiante es *candidato* a obtener una beca si está aprobado. Dados dos candidatos e_1 y e_2 diremos que e_1 es mejor que e_2 si e_1 tiene mejor nota que e_2, o si e_1 y e_2 tienen la misma nota y el DNI de e_1 es mayor que el DNI de e_2.
En todo momento el número de estudiantes con beca del conjunto `n_bec` será menor o igual que el número de becas disponibles `MAX_BEC`. Además `n_bec` nunca será superior al número de estudiantes aprobados `na`. Finalmente, si el número de estudiantes aprobados es mayor o igual que el número de becas disponibles `MAX_BEC`, el número de estudiantes con beca `n_bec` será igual a `MAX_BEC`.
Para implementar eficientemente esta funcionalidad hemos modificado la representación y la invariante de la clase `Estudiant` de la manera descrita en el archivo `Estudiant.hh`. En particular, representamos la información sobre si un estudiante tiene beca o no en los objetos de la clase `Estudiant`, y no en los objetos de clase `Cjt_estudiants`. Esto es, para asignar una beca al estudiante situado en la posición `pos` de `vest` hemos de utilizar la instrucción `vest[pos].modificar_beca(true);` y similarmente para comprobar si tiene beca o está aprobado. Leed con atención el archivo `Estudiant.hh`, especialmente las descripciones de los nuevos atributos, la invariante y las especificaciones de las operaciones nuevas. Las principales novedades de la clase `Estudiant` son:

- hemos incorporado un nuevo atributo `amb_beca` que permite representar información sobre si el estudiande parámetro implícito tiene beca o no;
- hemos añadido el consultor `te_beca` que permite comprobar si el estudiante parámetro implícito tiene beca o no;
- hemos añadido el modificador `modificar_beca` que permite modificar la información sobre si el estudiante parámetro implícito tiene beca o no;
- hemos añadido el consultor `aprovat` que permite comprobar si el estudiante parámetro implícito está aprobado o no;
- hemos añadido el método `static` y público `major_nota_dni` que permite comparar dos estudiantes por nota y en caso de empate por DNI.

También hemos modificado la representación y la invariante de la clase `Cjt_estudiants` de la manera descrita en el archivo `Cjt_estudiants.hh`. La principal novedad es que almacenamos la posición del mejor candidato no becado del conjunto en un nuevo atributo `i_max_no_becat`, de manera que si en algún momento hay una beca disponible conozcamos la posición del estudiante al cual hemos de asignar esta beca. Leed con atención el archivo

Cjt_estudiants.hh, especialmente las descripciones de los nuevos atributos, la invariante y las especificaciones de las operaciones nuevas. Las principales novedades de la clase Cjt_estudiants.hh son:

- hemos incorporado un nuevo atributo static y constante MAX_BEC que especifica el número de becas disponibles;
- hemos incorporado un nuevo atributo n_bec que contiene el número de estudiantes que tienen beca del parámetro implícito, que equivale al número de becas asignadas a estudiantes del p.i. de las MAX_BEC becas disponibles;
- hemos añadido un consultor b_assignades que permite consultar el valor del atributo n_bec;
- hemos incorporado un nuevo atributo i_max_no_becat que contiene la posición del mejor candidato no becado del parámetro implícito. Si hay candidatos no becados en el parámetro implícito, entonces i_max_no_becat contiene la posición del mejor candidato no becado y su valor está dentro del intervalo $0 \leq$ i_max_no_becat $<$ nest; en caso contrario, es decir, si no hay candidatos sin beca, el valor del atributo i_max_no_becat es igual a -1.
- hemos añadido el consultor público pos_max_no_becat, para consultar la posición del mejor candidato no becado. Si hay candidatos sin beca en el conjunto parámetro implícito devuelve i_max_no_becat+1, y si no hay candidatos sin beca devuelve -1.
- y hemos añadido el modificador privado recalcular_pos_max_no_becat para recalcular el valor del atributo i_max_no_becat cuando sea necesario.

Teniendo esto en cuenta debéis implementar eficientemente el siguiente método privado sin utilizar la operación sort de la biblioteca <algorithm>:

```
void recalcular_pos_max_no_becat();
/* Pre: cierto */
/* Post: Si hay candidatos no becados en el conjunto parámetro
implícito, el atributo i_max_no_becat contiene la posición del
mejor candidato no becado y su valor esta dentro del intervalo
0 <= i_max_no_becat < nest; si no hay candidatos sin beca, el
atributo i_max_no_becat es igual a -1. */
```

y el siguiente método público sin utilizar la operación sort de la biblioteca <algorithm>. Observad que cuando borramos un estudiante con beca del conjunto, su beca se asigna al mejor candidato no becado del conjunto, si hay alguno. Si el conjunto no contiene candidatos no becados, la beca del estudiante borrado no se asigna a ningún estudiante. Obviamente, si borramos un estudiante no becado, el número de estudiantes que tienen beca no se modifica.

```
void esborrar_estudiant(int x, bool& trobat);
/* Pre: cert  */
/* Post: Si el parámetro implícito original contenía un estudiante con
DNI = x, trobat es true, el p.i. contiene los mismos estudiantes que
el original menos el estudiante con DNI = x, se han actualizado los
estudiantes becados del p.i. si ha sido necesario, y se ha actualizado
la posición del mejor candidato no becado del p.i. si ha sido necesario;
en otro caso, trobat es false y el p.i. es igual al original. */
```

Observación

Debéis entregar un fichero `solucio.cc` con una implementación eficiente de las operaciones `recalcular_pos_max_no_becat` y `esborrar_estudiant` que ha de tener el siguiente formato:

```
#include "Cjt_estudiants.hh"

void Cjt_estudiants::recalcular_pos_max_no_becat() {
  ... // código de la implementación
}

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

Copiad esta plantilla en vuestro `solucio.cc` y completadla. Vuestro `solucio.cc` no puede contener la implementación de otras operaciones de la clase.
En el apartado *Public files* del Jutge os proporcionamos material adicional comprimido en un fichero .tar. Podéis descomprimir este fichero con el comando

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

Este material adicional contiene los siguientes ficheros:

- `Cjt_estudiants.hh`: la especificación Pre/Post de todas las operaciones públicas y privadas de esta nueva versión de la clase `Cjt_estudiants`, así como la definición de los atributos privados. Fijaos que **hemos añadido tres atributos** `n_bec, i_max_no_becat` y `MAX_BEC`, y que **hemos modificado la invariante** de la representación de `Cjt_estudiants`. **Es muy importante que la implementación de las operaciones que os hemos encargado tenga en cuenta y preserve la invariante de la representación**. Fijaos también que hay cuatro operaciones nuevas: el modificador privado `recalcular_pos_max_no_becat` y los consultores públicos `bec_disp, b_assignades, pos_max_no_becat`.

- `Cjt_estudiants.cc`: la implementación de todas las operaciones de la nueva versión de la clase `Cjt_estudiants` excepto las operaciones que os pedimos.

- `Estudiant.hh`: la especificación de la clase `Estudiant` y la definición de sus atributos. Las principales novedades que presenta son un atributo `amb_beca` que indica si se ha concedido una beca al estudiante parámetro implícito, y los métodos públicos `te_beca, modificar_beca, aprovat` y `major_nota_dni`.

- `Estudiant.cc`: la implementación de los métodos de la clase `Estudiant`.

- `pro2.cc`: un programa principal que podéis utilizar para probar los métodos públicos de esta versión de la clase `Cjt_estudiants`.

- `llegeixme.txt`: instrucciones para generar el ejecutable del programa `pro2` y probarlo.

Valoraremos positivamente que la solución no contenga instrucciones (especialmente bucles o llamadas a operaciones costosas) ni objetos (especialmente vectores o conjuntos) innecesarios, que no haga recorridos cuando debería hacer búsquedas, y que use correctamente las

operaciones más eficientes de la clase siempre que sea posible. No se puede usar ninguna estructura de datos que no haya aparecido en las sesiones 1-4 de laboratorio.

La utilización de la operación `sort` de la biblioteca `<algorithm>` en el archivo `solucio.cc` comportará una calificación de 0 en la corrección manual del control.

Cuando hagáis envíos, el Jutge os indicará cuantos juegos de pruebas pasa vuestro programa y de qué tipo (público o privado). El juego de pruebas denominado `público` corresponde a los ficheros `entrada.txt` y `sortida_correcta.txt` del apartado *Public files*.

Información del problema

Autor : Professors de PRO2
Traductor : Professors de PRO2
Generación : 2017-03-18 21:03:59