Classe multiconjunt d'enters X39772_ca

Cal implementar la següent classe *mcj_enters* que ens permet representar i manipular multiconjunts d'enters (no importa l'ordre dels enters i poden haver-hi enters repetits):

```
#include <iostream>
using namespace std;

class mcj_enters {

public:

  // Constructora per defecte. Crea un multiconjunt buit.
  mcj_enters();

  // Les tres grans: Constructora per còpia, destructora, operador d'assignació
  mcj_enters(const mcj_enters &cj);
  ~mcj_enters();
  mcj_enters& operator=(const mcj_enters &cj);

  // Insereix l'enter e en el multiconjunt.
  void insereix(int e);

  // Unió, intersecció i diferència de multiconjunts. Operen modificant el multiconjunt
  // sobre el que s'aplica el mètode, usant el segon multiconjunt com argument.
  // P.e.: a.restar(b)
  // fa que el nou valor d'a sigui A - B, on A i B són els valors originals dels objectes a i b.
  void unir(const mcj_enters& B);
  void intersectar(const mcj_enters& B);
  void restar(const mcj_enters& B);

  // Unió, intersecció i diferència de multiconjunts. Operen creant un nou multiconjunt
  // sense modificar el multiconjunt sobre el que s'aplica el mètode. La suma de
  // multiconjunts correspon a la unió, la resta a la diferència i el producte a la intersecció.
  mcj_enters operator+(const mcj_enters& B) const;
  mcj_enters operator*(const mcj_enters& B) const;
  mcj_enters operator-(const mcj_enters& B) const;

  // Retorna cert si i només si e pertany al multiconjunt.
  bool conte(int e) const;

  // Retornen els elements màxim i mínim del multiconjunt, respectivament.
  // El seu comportament no està definit si el multiconjunt és buit.
  int max() const;
  int min() const;

  // Retorna el nombre d'elements (la cardinalitat) del multiconjunt.
```

```
  int card() const;

  // Operadors relacionals. == retorna cert si i només si els dos multiconjunts
  // (el paràmetre implícit i B) contenen els mateixos elements;
  // != retorna cert si i només si els multiconjunts són diferents.
  bool operator==(const mcj_enters& B) const;
  bool operator!=(const mcj_enters& B) const;

  // Imprimeix el multiconjunt d'enters, ordenats en ordre ascendent, sobre
  // el canal de sortida os; el format és [e1 e2 ... en], és a dir, amb
  // espais entre els elements i tancant la seqüència amb corxets.
  void print(ostream& os) const;

private:
    // Cal definir els atributs i mètodes privats dins del fitxer .rep
    #include "mcj_enters.rep"
};
```

Bàsicament el que cal fer és:

1. Trobar una representació adequada pels objectes de la classe i escriure els atributs necessaris en la part *private* de la classe (dins del fitxer *mcj_enters.rep*). També es poden especificar mètodes privats addicionals.

2. Implementar tots els mètodes de la classe els quals manipularan la representació anterior dins del fitxer *mcj_enters.cpp*.

Cal enviar la solució (els fitxers *mcj_enters.rep* i *mcj_enters.cpp*) comprimida en un sol fitxer *.tar*
Per testejar la classe disposes d'un programa principal que processa blocs que contenen dos multiconjunts A i B i vàries comandes que els manipulen.

Entrada

L'entrada conté varis blocs separats per línies amb 10 guions (————). Cada bloc consisteix en dues seqüències d'enters, una per línia, cadascuna d'elles són els elements que tindran originalment el multiconjunt A i el multiconjunt B. A continuació segueixen vàries comandes, una per línea, amb el següent format:

- insereix cjt1 e
- conte cjt1 e
- max cjt1
- min cjt1
- card cjt1
- unir cjt1 cjt2
- intersectar cjt1 cjt2

- restar cjt1 cjt2
- + cjt1 cjt2
- * cjt1 cjt2
- - cjt1 cjt2
- == cjt1 cjt2
- !=cjt1 cjt2
- print cjt1

On cjt1 i cjt2 poden ser 'A' o 'B'.

Sortida

Per a cada línia d'entrada, escriu una línia amb el resultat:

- Si la línia és un multiconjunt, mostra el multiconjunt un cop inserit tots els seus elements.
- Si la línia és una comanda, mostra la comanda, el separador ": " i el resultat. Si la comanda retorna o modifica algun multiconjunt, mostra aquest multiconjunt.
- Si la línia és el separador de blocs format per 10 guions, mostra els mateixos 10 guions.

Observació

Aquest problema proporciona la definició pública de la classe *mcj_enters* dins del fitxer *mcj_enters.hpp*, el programa principal *main.cpp* i un fitxer *Makefile* per facilitar la compilació.
Per implementar el multiconjunt no es poden usar les classes *stack*, *queue*, *list* o *set* de la STL.
Pots fer un parell de versions, una implementada amb memòria estàtica (usant per exemple arrays de C++) i una altra implementada amb memòria dinàmica. Si la versió amb memòria estàtica té un límit en el màxim nombre d'element d'un multiconjunt, segurament no passarà els jocs de prova privats on hi ha casos amb multiconjunts molt grans.
Si els mètodes d'unir, intersectar, restar, igualtat i diferència no es programen de forma eficient (cost lineal) tampoc passaran els jocs de prova privats degut a un excés de temps d'execució.

Exemple d'entrada 1

```
1
conte A 0
conte A 1
conte B 0
conte B 1
max B
min B
card A
card B
----------
4 -8 12 -6 14 0
1 -5 13 12 -8
```

```
conte A 0
conte A 1
conte B 0
conte B 1
max A
min A
max B
min B
card A
card B
```

Exemple de sortida 1

```
[]
[1]
conte A 0: 0
conte A 1: 0
conte B 0: 0
conte B 1: 1
max B: 1
min B: 1
card A: 0
card B: 1
----------
[-8 -6 0 4 12 14]
[-8 -5 1 12 13]
conte A 0: 1
conte A 1: 0
conte B 0: 0
conte B 1: 1
max A: 14
min A: -8
max B: 13
min B: -8
card A: 6
card B: 5
```

Exemple d'entrada 2

```
1
insereix B 1
conte B 0
conte B 1
max B
min B
card B
insereix B 0
conte B 0
conte B 1
max B
min B
card B
+ A A
* A A
- A A
== A A
!= A A
+ B B
* B B
- B B
== B B
!= B B
+ A B
* A B
- A B
- B A
== A B
!= A B
unir A B
intersectar A B
restar A B
----------
2 1 3 1
4 2 1 2
+ A A
* A A
- A A
== A A
!= A A
+ B B
* B B
- B B
== B B
```

```
!= B B
+ A B
* A B
- A B
- B A
== A B
!= A B
unir A B
intersectar A B
restar A B
----------
4 -8 12
12 -8 4
== A B
!= A B
```

Exemple de sortida 2

```
[]
[1]
insereix B 1: [1 1]
conte B 0: 0
conte B 1: 1
max B: 1
min B: 1
card B: 2
insereix B 0: [0 1 1]
conte B 0: 1
conte B 1: 1
max B: 1
min B: 0
card B: 3
+ A A: []
* A A: []
- A A: []
== A A: 1
!= A A: 0
+ B B: [0 1 1]
* B B: [0 1 1]
- B B: []
== B B: 1
!= B B: 0
+ A B: [0 1 1]
* A B: []
- A B: []
- B A: [0 1 1]
== A B: 0
!= A B: 1
unir A B: [0 1 1]
intersectar A B: [0 1 1]
restar A B: []
----------
[1 1 2 3]
[1 2 2 4]
+ A A: [1 1 2 3]
* A A: [1 1 2 3]
- A A: []
== A A: 1
!= A A: 0
+ B B: [1 2 2 4]
* B B: [1 2 2 4]
- B B: []
== B B: 1
!= B B: 0
+ A B: [1 1 2 2 3 4]
* A B: [1 2]
- A B: [1 3]
- B A: [2 4]
== A B: 0
!= A B: 1
unir A B: [1 1 2 2 3 4]
intersectar A B: [1 2 2 4]
restar A B: []
----------
[-8 4 12]
[-8 4 12]
== A B: 1
!= A B: 0
```

Informació del problema

Autoria: Jordi Esteve

Generació: 2026-01-25T21:11:20.765Z