Biliardo - testo

Su un Biliardo sono disposte sfere numerate. Le sfere disposte su uno stesso biliardo hanno numeri *diversi*. Le operazioni che possono essere effettuate su un biliardo sono le seguenti:

- **Biliardo d()**
  - **Costruttore di default, che inizializza un biliardo d. Inizialmente, su tale biliardo non sono disposte sfere.**
- **Biliardo d(n)**
  - **Costruttore di conversione, che consente di usare un int ovunque occorre un biliardo. Il costruttore inizializza il biliardo d. Inizialmente, su tale biliardo è disposta una sola sfera, e tale sfera ha numero n.**
- **Biliardo d1(d)**
  - **Costruttore di copia, che inizializza un biliardo d1 col valore del biliardo d.**
- **d1 = d**
  - **Operatore di assegnamento, che sostituisce il valore del biliardo risultato d1 con quello del biliardo d.**
- **d += i**
  - **Operatore di somma e assegnamento, che, se necessario, aggiunge una nuova sfera al biliardo d, in modo tale che il numero i sia presente su una delle sfere disposte su d.**
- **d1 += d2**
  - **Operatore di somma e assegnamento, che, se necessario, aggiunge nuove sfere al biliardo d1, in modo tale che ciascuno dei numeri delle sfere disposte sul biliardo d2 sia anche presente su una delle sfere disposte su d1.**

Biliardo- testo

- **d -= i**
  - **Operatore di sottrazione e assegnamento, che, se necessario, toglie una sfera dal biliardo d, in modo tale che il numero i non sia presente sulle sfere disposte su d.**
- **d1 -= d2**
  - **Operatore di sottrazione e assegnamento, che, se necessario, toglie sfere dal biliardo d1, in modo tale che nessuno dei numeri delle sfere disposte sul biliardo d2 sia presente sulle sfere disposte su d1.**
- **cout << d**
  - **Operatore di uscita per il tipo Biliardo. L'uscita consiste nei numeri delle sfere disposte sul biliardo d, in ordine crescente, separati da virgole e racchiusi tra parentesi angolate. Esempio:**
  - **<1, 3, 5, 6>**
- **~Biliardo()**
  - **Distruttore.**

Mediante il linguaggio C++, realizzare il tipo di dati astratti Biliardo, definito dalle precedenti specifiche. *Utilizzare una rappresentazione del biliardo a lista.* Individuare le eventuali situazioni di errore, e metterne in opera un corretto trattamento.

Biliardo.h

```
#include <iostream.h>

class Biliardo {
friend ostream& operator<<(ostream&, const Biliardo&);
	struct elem {
		int numero;
		elem* pun;
	};
	elem* testa;
	void copia(const Biliardo&);
	void elimina();
public:
	Biliardo() {testa = NULL;}
	Biliardo(int);
	Biliardo(const Biliardo& d) {copia(d);}
	Biliardo& operator=(const Biliardo&);
	Biliardo& operator+=(const Biliardo&);
	Biliardo& operator+=(int);
	Biliardo& operator-=(const Biliardo&);
	Biliardo& operator-=(int);
	~Biliardo() {elimina();}
};

void Biliardo::copia(const Biliardo& d) {
	testa = NULL;
	if (d.testa != NULL) {
		testa = new Biliardo::elem;
		testa->numero = d.testa->numero;
		Biliardo::elem* aux = d.testa->pun;
		Biliardo::elem* tmp = testa;
		while (aux != NULL) {
			tmp->pun = new Biliardo::elem;
			tmp = tmp->pun;
			tmp->numero = aux->numero;
			aux = aux->pun;
		}
		tmp->pun = NULL;
	}
}
```

Biliardo.cpp

```
void Biliardo::elimina() {
	Biliardo::elem* aux = testa;
	while (testa != NULL) {
		testa = testa->pun;
		delete aux;
		aux = testa;
	}
}

Biliardo::Biliardo(int i) {
		testa = new Biliardo::elem;
		testa->numero = i;
		testa->pun = NULL;
}

Biliardo& Biliardo::operator=(const Biliardo& d) {
	if (this != &d) {
		elimina();
		copia(d);
	}
	return *this;
}

Biliardo& Biliardo::operator+=(int i) {
	Biliardo::elem *p, *q, *r;
	for (r = testa; r != NULL && r->numero < i; r = r->pun)
		p = r;
	if (r != NULL && r->numero == i)
		return *this;
	q = new Biliardo::elem;
	q->numero = i;
	q->pun = r;
	if (r == testa)
		testa = q;
	else
		p->pun = q;
	return *this;
}
```

Biliardo.cpp

```
Biliardo& Biliardo::operator+=(const Biliardo& d) {
	Biliardo::elem* p = d.testa;
	while (p != NULL) {
		(*this)+=(p->numero);
		p = p->pun;
	}
	return *this;
}


Biliardo& Biliardo::operator-=(int i) {
	Biliardo::elem *p, *r;
	for (r = testa; r != NULL && r->numero < i; r = r->pun)
		p = r;
	if (r == NULL || r->numero > i)
		return *this;
	if (r == testa)
		testa = testa->pun;
	else
		p->pun = r->pun;
	delete r;
	return *this;
}

Biliardo& Biliardo::operator-=(const Biliardo& d) {
	Biliardo::elem* p = d.testa;
	while (p != NULL) {
		(*this)-=(p->numero);
		p = p->pun;
	}
	return *this;
}
```

Biliardo.cpp

```
ostream& operator<<(ostream& os, const Biliardo& d) {
	os << '<';
	if (d.testa != NULL) {
		os << d.testa->numero;
		Biliardo::elem* aux = d.testa->pun;
		while (aux != NULL) {
			os << ", ";
			os << aux->numero;
			aux = aux->pun;
		}
	}
	os << '>';
	return os;
}
```