SpotErrors

Table of contents

Code

La classe templatica **`Set`** è intesa rappresentare un insieme di elementi di tipo `T`. L'implementazione della classe si basa sulla manipolazione di liste ordinate (senza duplicati). L'interfaccia della classe presenta numerosi problemi. Cercare d'individuarne il maggior numero e indicare come possono essere risolti (riscrivendo l'interfaccia).

```
template <typename T>
class Set : public std::list<T> { // class Set : private std::list<T> {
        public:
                Set();
                Set(T t); // explicit Set(const T& t)
                Set(Set y); // Set(const Set& y)
                void operator=(Set y); // void operator=(const Set& y)
                virtual ~Set(); // ~Set();

                unsigned int size(); // unsigned int size() const
                bool is_empty(); // bool is_empty() const
                bool contains(Set y); // bool contains(const Set& y) const

                T& min(); // const T& min() const
                void pop_min();
                void insert(T z); // void insert(const T& z)
                void union_assign(Set y); // void union_assign(const Set& y)
                void intesection_assign(Set y);
                        // void intersection_assign(const Set& y)

                void swap(Set y); // void swap(Set& y)
                std::ostream operator<<(std::ostream os);

        private:
                // ...
}
// template <typename T>
// std::ostream& operator<<(std::ostream& os, const Set<T>& t) const
```

Comments

- Il costruttore dell'insieme vuoto della classe è ben formato
- Il costruttore elemento in `Set` meglio non abbia conversione implicita, preferiamo passare per rifermento evitando copie di `T` molto grandi
- Il costruttore di copia non necessita di essere marcato `explicit`, prende un `const`
- Il distruttore dichiarato `virtual` impedisce di essere invocato da altre classi e quindi distruggere l'oggetto
- Le funzioni `is_empty()` e `contains()` necessitano `const` per promettere di non modificare il parametro in ingresso
- La funzione `min()` può avere due possibili forme:
    - nel caso mi venga restituito un riferimento al valore, allora questa ha il solo scopo di trovare l'elemento minimo nella lista senza farci nulla;

    - nel caso volessimo modificare il tipo di ritorno, dovremmo prima considerare il concetto d'*invariante di classe* che verrebbe violata dall'utente (questo è il caso)
- La funzione `pop_min()` ha l'obbiettivo di fare "pop" sull'elemento minimo della lista
- La funzione `insert()` necessita `const`
- `union_assign()` con `const`
- `intersection_assign()` con `const`
- `swap()` modifica i riferimenti dei contenuti che vogliamo scambiare, quindi passiamo un riferimento
- L'operatore di output va messo fuori dalla dichiarazione della classe, e modificato in modo tale da poter permettere concatenzazione degli output; necessita inoltre dei `const` dei parametri
- Tutti i metodi della classe base di `List` sono inclusi all'interno di `Set`, perché eredita pubblicamente la classe; possono essere applicati tutti i metodi delle liste e l'invariante può essere violata molto semplicemente. Notare come nella consegna dell'esercizio, ci viene chiesto che `Set` sia usata per rappresentare.