21 - Programmazione Generica (lez. 27-11-2024)

Contenitori Associativi

Contenitori che organizzano gli elementi al proprio interno in modo da facilitarne la ricerca (non in base alla posizione) in base al valore di una chiave.
I contenitori associativi della libreria standard si differenziano in base a tre caratteristiche binarie:

- `set<Key, Cmp>` //ordered è implicito dove non è scritto
- `multiset<Key, Cmp>`
- `map<Key, Mapped, Cmp>`
- `multimap<Key, Mapped, Cmp>`
- `unordered_set<Key, Hash, Equal>`
- `unordered_multiset<Key, Hash, Equal>`
- `unordered_map<Key, Mapped, Hash, Equal>`
- `unordered_multimap<Key, Mapped, Hash, Equal>`

Caratteristiche Binarie

1. Presenza di informazioni supplementari, oltre la chiave. (es. key + altro: `map`, solo key: insieme);
2. possibilità di memorizzare più elementi con lo stesso valore per la chiave (es. versioni `multi`);
3. struttura dati usata per il contenitore (modalità di implementazione interna, sfruttano un criterio di ordinamento sulle chiavi), esempio:
   - le versioni `sorted` utilizzano alberi di ricerca bilanciati, basati su un criterio di ordinamento(`Cmp`) definito sul tipo `Key`;
   - le versioni `unsorted` utilizzano le tabelle hash, basate su una funzione di hashing `Hash` e una relazione equivalente `Equal` per risolvere i conflitti di chiave.

```cpp
#include <iostream>
#include <map>
#include <iterator>
#include <string>

//es: programma che legge dallo standard input delle stringhe e conta quante volte quella stringa compare
int main() {
    std::map<std::string, unsigned long> freq_map; //chiave stringa, valore quante volte compare la stringa
    std::istream_iterator<std::string> i_first(std::cin);
    std::istream_iterator<std::string> i_last;

    for ( ; i_first != i_last; ++i_first) {
        const auto& s = *i_first; //stringa s ottenuta deferenziado i_first
        //voglio controllare se s è nella mappa, se non c'è la inserisco la inserisco incrementando il contatore a uno, se già è presente incremento il contatore senza inserire di nuovo
        //se non forniamo noi il criterio di ordinamento quello di default è l'operator< sulla chiave (in questo caso in base alla string -> ordine alfabetico)
```

```
        //METODO NORMALE: trova se esiste nel contenitore un elem con questa chiave: .find(str);
        auto it = freq_map.find(s); //restituisce un iteratore che punta all'elemento, se punta all'ultimo elemento del contenitore, vuol dire che l'elem non è stato trovato
        if (it == freq_map.end()) {
            //inserisci e incrementa
            //freq_map.insert(std::pair<std::string, unsigned long>(s,1))
            freq_map.insert(std::make_pair(s, 1));
        } else {
            //incrementa
            ++(it->second); //++(*it).second
        }
    }
    //METODO SMART: possiamo usare [] come per gli array, ma invece di un indice numerico ci mettiamo il valore della chiave
    for ( ; i_first != i_last; ++i_first) {
        const auto& s = *i_first;
        if (it == freq_map.end()) {
            //freq_map.insert(std::make_pair(s, 1));
            ++freq_map[s]; //come valore mette il valore che si ottiene intuitivamente con il costruttore di default del tipo mappato: unsigned long() value initialization -> 0, ma noi vogliamo 1, quindi incrementiamo
        } else {
            ++freq_map[s]; //l'operator[] quando non trova l'elem lo inserisce automaticamente
        }
    }
    //VERO METODO SMART
    for ( ; i_first != i_last; i_first++){
        const auto& s = *i_first;
        ++freq_map[s];
    }

    for (const auto& p : freq_map) {
        std::cout << "La parola " << p.first
        << " occorre numero " << p.second << " volte\n";
    }
}
```