1 Llistes

2 Sets

3 Maps

Sets

La classe `set` representa conjunts d'elements ordenats (respecte a una relació <) y sense repeticions.
Els elements del set no poden ser modificats, però sí poden ser insertats o eliminats del set.
Totes les operacions de set tenen cost logarítmic.

```
#include <iostream>
#include <string>
#include <set>

using namespace std;

int main(){
  set<string> s;
  s.insert("Laura");   // s: {Laura}
  s.insert("Guille"); // s: {Guille,Laura}
  s.insert("Sonia");   // s: {Guille,Laura,Sonia}
  s.insert("Nuria");   // s: {Guille,Laura,Nuria,Sonia}
  s.erase("Laura");    // S: {Guille,Nuria,Sonia}
  cout << s.count("Guille") << endl; // output: 1
  cout << s.count("Manel") << endl;  // output: 0
}
```

Especificació de la classe `Set`

```
template <class T> class set {
public:

// Subclasses de la classe set
     class iterator { ... };
     class const_iterator { ... };

// Constructores

/* Pre: cert */
/* Post: El resultat es un conjunt sense cap element */
set();

// Modificadores
/* Pre: cert */
/* Post: el conjunt implícit queda buit */
void clear();
```

Especificació de la classe genèrica `Set`

Podem utilitzar la clase iterator i la const_iterator sobre sets.

Més modificadores:

```
/* Pre: cert */
/* Post: al conjunt implícit se li afegeix l'element x */
void insert(const T& x);

/* Pre: it referencia algun element existent al set */
/* Post: s'ha eliminat l'element referenciat per it,
i retorna un iterador al sucessor de l'element eliminat
o a end() si era l'únic */
iterator erase(iterator it);

/* Pre: cert */
/* Post: s'ha eliminat l'element x si hi era, i retorna 1
si l'element s'ha eliminat i 0 si no */
int erase(const T& x);
```

Sets

Consultores:

```
/* Post: retorna un iterador a la posició on es troba x
o a end si no es troba */
iterator find(const T&x) const;

// Post: retorna el tamany del set paràmetre implícit
int size() const;

// Post: retorna si el paràmetre implícit es buit o no
bool empty() const;

// Post: retorna 1 o 0 si l'element x està o no al p.i.
int count(const T& x);

// tornen iteradors al primer element del set
const_iterator begin() const;
iterator begin();

// tornen iteradors a l'element fictici succesor de l'últim
 const_iterator end() const;
 iterator end();
```

Escriure sets

Utilització de iterator.

També hi ha un element fictici al final del set, a la posició end.

```
void printSet(const set<string> &s)
{
  cout << "{";
  bool printcomma = false;
  set<string>::const_iterator it;
  for(it = s.begin(); it != s.end(); it++){
    if (printcomma) cout << ", ";
    printcomma = true;
    cout << *it;
  }
  cout << "}" << endl;
}
```

També podem utilitzar la capçalera
void printSet(set<string> s)
i llavors podem utilitzar un iterator en lloc d'un const_iterator.

1 Llistes

2 Sets

3 Maps

Maps

- Un diccionari o map és una seqüencia de parells clau$\rightarrow$valor.
- Estructura amb templates. Tant el tipus de les claus com el tipus de valors poden ser qualsevols.
- Els parells estan ordenats de forma creixent per la clau. Ha de haver una relació d'ordre $<$ definida.
- Les operacions d'aquesta clase s'executen en temps com a molt logarismic.
- Podem utilitzar la notació vector, o la notació list.
- Podem utilitzar iterators i const_iterators.

Map. Notació vector

Indexar amb un tipus de dades que tingui (o poguem definir) <.

La instrucció age[“Laura"]=42 crea el parell Laura:42 i l’inserta al map, o si Laura ja està al map canvia el seu valor.

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

int main(){
  map<string,int> age;
  age["Laura"] = 42;    // {Laura:42}
  age["Guille"] = 50;   // {Guille:50,Laura:42}
  age["Sonia"] = 16;    // {Guille:50,Laura:42,Sonia:16}
  cout << age["Sonia"] << endl;  // output: 16
  cout << age["Guille"] << endl; // output: 50
  age["Sonia"] = 17;       // {Guille:49,Laura:42,Sonia:17}
  age["Guille"]++;      // {Guille:51,Laura:42,Sonia:17}
//accès a posició no definida la inicialitza al valor per defecte
  age["Nuria"]++;       // {Guille:51,Laura:42,Nuria:1,Sonia:17}
}
```

Map. Notació list

Modificadoras: insert, erase.

```
map<string,int> age;
age.insert(make_pair("Jose",50));     // {Jose:50}
age.insert(pair<string,int>("Ana",36));  // {Ana:36,Jose:50}

age.insert(make_pair("Ana",20));      // {Ana:36,Jose:50}
//si insertem Ana un altre cop, no canvia el map

pair<map<string,int>::iterator,bool> p;
p=age.insert(make_pair("Maria",36)); // {Ana:36,Jose:50,Maria:36}
// retorna un iterator que apunta a l'element amb clau Maria
// el booleà indica si s'ha pogut introduir el parell

map<string,int>::iterator it;
it=age.erase(p.first);                  // {Ana:36,Jose:50}
//l'iterator apunta a l'element després de l'apuntat per p.first.
//en aquest cas apunta a age.end()

int x=age.erase("Ana");               // {Jose:50}
//retorna 0 si Ana no hi era
```

Map

Consultores:

```
/* retorna un iterador a la posició on es troba x
o a end si no es troba */
iterator find(const T&x) const;

/* Pre: cert */
/* Post: retorna el tamany del map paràmetre implícit */
int size() const;

/* Pre: cert */
/* Post: retorna si el paràmetre implícit es buit o no */
bool empty() const;

/* Pre: cert */
/* Post: retorna 1 o 0 si l'element x està o no al p.i. */
int count(const T& x);

// tornen iteradors al primer element del map
const_iterator begin() const;
iterator begin();

// tornen iteradors a l'element fictici succesor de l'últim
 const_iterator end() const;
 iterator end();
```

Aspectes subtils de maps

Podem modificar un valor d'un map, però no podem modificar una clau.

```
map<string,int>::iterator it=age.begin();
it->second +=10;      // correcte {Jose:60}
it->first= "Maria"    // INCORRECTE!!!
```

Diferencia entre age.find("Pep") i age["Pep"]:
El primer retorna un iterator a end. El segon crea el parell Pep:0 i l'inserta.

```
void consulta(map<string,int> &age){
   cout<< "Albert"<<" "<< age["Albert"]<<endl;
}
```

En aquest cas la capçalera
void consulta(const map<string,int> &age) es incorrecta.

Escriure maps

```
void printMap(const map<string,int> &m)
{
  cout << "{";
  bool printcomma = false;
  map<string,int>::const_iterator it;
  for(it = m.begin();it!= m.end();it++){
    if (printcomma) cout << ", ";
    printcomma = true;
    cout << it->first << ":" << it->second;
  }
  cout << "}";
  cout << endl;
}
```