LECCION 16: STL

Conjunto (Set): Son contenedores que almacenan elementos no repetidos de un tipo base de una forma ordenada. El valor de un elemento en el conjunto se denota como llave (Key).

Funciones miembros

- count: busca que elementos son iguales a un valor de entrada y devuelve el nº de iguales

ej:

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

int main(){
  set<int> micanj;
  for(int i=1; i<5; i++) micanj.insert(i*3); //set: 3 6 9 12

  for (int i=0; i<10; i++){
        cout<<i;
        if (micanj.count(i)!=0){//está en el conjunto
           cout<<"esta en el conjunto";
        }
        else cout<<"no esta"<<endl;

  }
}
```

- swap: intercambia el contenido del contenedor por otro.

ej:

```
#include <iostream>
#include <set>
using namespace std;
int main(){
   int myv[]={12,75,10,32,20,25};
    set<int> first(myv, myv+3); // 10 12 75
    set<int> second(myv+3, myv+6); // 20 25 32
    first.swap(second);

}
```

(2) LECCION 16: STL

• find: busca en el conjunto un elemento y devuelve un iterador apuntando a él. Si no se encuentra el iterador apunta a end().

Ej:
```
#include <iostream>
#include <set>
using namespace std;
int main(){
  set<int> miconj;
  for(int i=1; i<=5; i++) miconj.insert(i*10); // 10 20 30
                                               // 40 50

  set<int>::iterator it = miconj.find(20);
  miconj.erase(it);
  miconj.erase(miconj.find(40));
  for(it=miconj.begin(); it!=miconj.end(); ++it)
        cout << *it << " ";

}
```

• equal_range

pair<iterator, iterator> equal_range(const value_type &val)const

- devuelve dos iteradores. El primero apunta al primer elemento que tiene el mismo valor que el parámetro de entrada. El segundo iterador apunta al primer elemento a continuación que ya no coincide con el parametro de entrada.

```
#include <iostream>
#include <set>
using namespace std;
int main(){
  set<int> miconj;
  for(int i=1; i<=5; i++) miconj.insert(i*10); // 10 20 30 40 50

  pair<set<int>::const_iterator, set<int>::const_iterator> ret;

  ret = miconj.equal_range(30);
  cout << "Limite inferior: " << *ret.first << endl; // 30
  cout << "Limite superior: " << *ret.second << endl; // 40
}
```

LECCION 16 : STL

upper_bound: devuelve un iterador apuntando a el primer elemento mayor que el valor de entrada

lower_bound: devuelve un iterador apuntando a el primer elemento igual (si existe) o menor que el valor de entrada.

Ej:

```
#include <iostream>
#include <set>
 using namespace std;
 int main(){
   set <int> micanj;
   set <int>::iterator itlow,itup;
   for(int i=1; i<10; i++) micanj.insert(i*10); //10 20 30
                                                //40 50 60
                                                //70 80 90

   itlow = micanj.lower_bound(30);
   itup = micanj.upper_bound(60);
   micanj.erase(itlow,itup);

 }
```

- value_comp: devuelve un objeto comparador de set que se puede usar para comparar dos objetos del contenedor. Devuelve true si el primer elemento es menor que el segundo.

ej: Funcion para contabilizar cuantos objetos son menores que un valor dado

```
int menores(const set<int> &s, int x){
   set<int>::value_compare micomp = s.value_comp(); //s.key_comp()
   set<int>::const_iterator it= s.begin();
   int cnt=0;
   while(micomp(*it, x)){
          cnt++;
          ++it;
   }
   return cnt;

}
```

4 LECCION 16 : STL

ejercicio

```
template <class T>
class Conjunto {
    private:
        list <T> datos;

    public:
    pair < typename list<T>::iterator, bool> Esta ( const T& v) const{
        typename list<T>:: const_iterator it_low = datos.begin(),
                                          it_high = datos.end() ;

        int n=datos.size();
        while ( n >= 1) {
            typename list<T> :: const_iterator mitad= it_low;
            advance (mitad, n/2);
            if ( *mitad == v) {
                pair < typename list<T>::iterator, bool> p(mitad, true);
                return p;
            }
            else
                if (*mitad < v) {
                    it_low = mitad + 1;  n =(n - n/2) - 1;

                }
                else{
                    it_high = mitad - 1;
                    n = n/2 - 1;

                }
        }
        pair < typename list <T> :: iterator, bool > p (it_low, false);
        return p;
    }
```

4 LECCIÓN 16 : STL

ejercicio Conjunto (continuación)

```
void Insertar ( const T &v) {
pair<typename list<T>::const_iterator,bool> p;
        p=Esta(v);
        if (p.second==false)
            datos.insert ( p.first, v );

}
void Borrar (const T &v) {
    pair <typename list <T>::const_iterator,bool> p;
     p=Esta(v);
     if ( p.second )
            datos.erase ( p.first);

}
Conjunto <T> operator + ( const Conjunto <T> & C) {
      Conjunto <T> Caux (*this);
      typename list<T>::const_iterator it_c;
      for (it_c=C.datos.begin(); it_c!=C.datos.end(); ++it_c){
                  Caux.Insertar( *it_c);

      }return Caux;
}
```