LECCIÓN 13

Abstracción por iteración

Contenedores — son estructuras de datos que mantienen almacenadas una colección de elementos de otro tipo de dato.

⇓

¿Forma de acceder a los elementos de los contenedores?

⇓

ITERADORES (≃ TDA POSICION)

└ TDA que abstrae la idea de acceder a los elementos de un contenedor como una forma parecida a los punteros.

COMO USARLOS

1.- Iniciar el iterador a la primera posición ⟹ begin

2.- Saber como avanzar (++, --, ...)

3.- Se debe saber como acceder al elemento que apunta (*)

4.- Saber cuando terminar

(Ver transparencia de Lista)

```
Lista l
  :
Lista:: iterator it;
for (it= l.begin(); it!=l.end(); ++it)
        *it = (*it) +1;
```

- Necesidad de iteradores constantes

```
void Imprimir_lista (const Lista &L)
{
   Lista:: const_iterator it;
   for (it=L.begin(); it!=L.begin()
        ++it)
         cout << *it;
}
```

Transparencia 2

Ejercicio: crear una clase Notas en la que se almacena pares dni, nota. Para almacenar el conjunto de pares usaremos la clase vector de la STL. Definir dentro de ella la clase iterator y la clase const_iterator. Además:

1) Crear una función que imprima los pares

2) Crear una función para obtener la nota media

3) Crear una función para modificar a un alumno su nota.

```
class Lista{
 private:
  int *datos;
  int n; //elementos almacenados
  int reservados;

 public:
  ……
//DOS ITERADORES
class const_iterator;
 class iterator{
 private :
  int * it;
 public:
    iterator(){}
    bool operator==(const iterator &i)const{
          return i.it==it;
    }
    bool operator!=(const iterator & i)const{
          return i.it!=it;
    }
    int & operator *(){
          return *it;
    }
    iterator & operator++(){
          ++it;
          return *this;
    }
    iterator & operator--(){
          --it;
          return *this;
    }
    friend class Lista;
    friend class const_iterator;
}; //fin iterator

 class const_iterator{
   private:
    const int *it;
   public:
     const_iterator(){}
     const_iterator(const iterator &i):it(i.it){}
     bool operator==(const const_iterator
&i)const{
          return i.it==it;
     }
     bool operator!=(const const_iterator & i)const{
          return i.it!=it;
     }
     const int & operator *()const{
          return *it;
     }
     const_iterator & operator++(){
          ++it;
          return *this;
     }
     const_iterator & operator--(){
          --it;
          return *this;
     }
     friend class Lista;
 }; //fin const_iterator
 iterator begin(){
   iterator i;
   i.it= &(datos[0]);
   return i;
 }
 iterator end(){
   iterator i;
   i.it = &(datos[n]);
  return i;
}

iterator begin(){
     iterator i;
     i.it= &(datos[0]);
     return i;
 }
 iterator end(){
     iterator i;
    i.it = &(datos[n]);
   return i;
}
const_iterator begin() const{
     const_iterator i;
     i.it= &(datos[0]);
     return i;
 }
const_iterator end()const {
    const_ iterator i;
    i.it = &(datos[n]);
   return i;
}
};
```

```
#ifndef __NOTAS__H
#define __NOTAS__H
1.  #include <vector>
2.  #include <string>
3.  #include <iostream>
4.  using namespace std;
5.
6.  class Notas{
7.  private:
8.    vector<pair<string,float> > datos;
9.  public:
10.  Notas(){}
11.  pair<string,float >& operator[](int i){ return
     datos[i];}
12.  const pair<string,float >& operator[](int i)const
     { return datos[i];}
13.  int size()const{   return datos.size();}
14.  friend istream & operator >>(istream &is, Notas
     &N);
15.  class const_iterator; //declaracion adelantada
16.
     //Clases Iteradores
17.  class iterator{
18.    private:
19.      vector<pair<string,float> >::iterator it;
20.  public:
21.      iterator(){}
22.      bool operator==(const iterator &i)const{
23.            return i.it==it;
24.      }
25.      bool operator!=(const iterator & i)const{
26.           return i.it!=it;
27.      }
28.      pair<string,float> & operator *(){
29.           return *it;
30.      }
31.      iterator & operator++(){
32.           ++it;
33.           return *this;
34.      }
35.      iterator & operator--(){
36.           --it;
37.           return *this;
38.      }
39.      friend class Notas;
40.      friend class const_iterator;
41.  }; //fin iterator
42.
43.  class const_iterator{
44.    private:
45.     vector<pair<string,float> >::const_iterator it;
46.    public:
47.     const_iterator(){}
48.     const_iterator(const iterator &i):it(i.it){}
49.     bool operator==(const const_iterator &i)const{
50.           return i.it==it;
51.     }
52.     bool operator!=(const const_iterator & i)const{
53.           return i.it!=it;
54.     }
55.     const pair<string,float> & operator *()const{
56.           return *it;
57.     }
58.     const_iterator & operator++(){
59.           ++it;
60.           return *this;
61.     }
62.     const_iterator & operator--(){
63.           --it;
64.           return *this;
65.     }
66.     friend class Notas;
67.  }; //fin const_iterator
68.
69.  iterator Insertar(iterator i,const
     string &dni,float nota){
70.       pair<string,float>
     a(dni,nota);
71.           iterator iter;
72.       iter.it=datos.insert(i.it,a);
73.           return iter;
74.  }
75.  iterator Borrar(iterator i){
76.     iterator iter;
77.     iter.it=datos.erase(i.it);
78.     return iter;
79.  }
80.
81.  iterator begin(){
82.     iterator i;
83.     i.it=datos.begin();
84.     return i;
85.  }
86.  iterator end(){
87.     iterator i;
88.     i.it=datos.end();
89.     return i;
90.  }
91.  const_iterator begin()const{
92.     const_iterator i;
93.     i.it=datos.begin();
94.     return i;
95.  }
96.  const_iterator end()const{
97.     const_iterator i;
       i.it=datos.end();
98.  return i;
     }
     #endif
```

```
#ifndef __NOTAS__H
#define __NOTAS__H
#include <vector>
#include <string>
#include <iostream>
using namespace std;

class Notas{
private:
   vector<pair<string,float> > datos;


public:
  Notas(){}

  pair<string,float >& operator[](int i){ return datos[i];}

  const pair<string,float >& operator[](int i)const { return datos[i];}
  int size()const{
     return datos.size();
  }

  friend istream & operator >>(istream &is, Notas &N){
      N.datos.clear();
      string dni;
      float nota;
      while ((is>>dni>>nota)){
           pair<string,float> a(dni,nota);
           N.datos.push_back(a);
      }
      return is;
  }

  class const_iterator;

  class iterator{
    private:
      vector<pair<string,float> >::iterator it;
  public:
       iterator(){}
       bool operator==(const iterator &i)const{
           return i.it==it;
       }
       bool operator!=(const iterator & i)const{
          return i.it!=it;
       }
       pair<string,float> & operator *(){
          return *it;
       }
       iterator & operator++(){
          ++it;
          return *this;
       }
       iterator & operator--(){
          --it;
          return *this;
       }
       friend class Notas;
       friend class const_iterator;
  };
  class const_iterator{
     private:
```



```
       vector<pair<string,float> >::const_iterator it;
     public:
       const_iterator(){}
       const_iterator(const iterator &i):it(i.it){}
       bool operator==(const const_iterator &i)const{
           return i.it==it;
       }
       bool operator!=(const const_iterator & i)const{
          return i.it!=it;
       }
       const pair<string,float> & operator *()const{
          return *it;
       }
       const_iterator & operator++(){
          ++it;
          return *this;
       }
       const_iterator & operator--(){
          --it;
          return *this;
       }
       friend class Notas;
  };

  iterator  Insertar(iterator i,const string &dni,float nota){
        pair<string,float> a(dni,nota);
        iterator iter;
        iter.it=datos.insert(i.it,a);
        return iter;
  }
  iterator Borrar(iterator i){
      iterator iter;
      iter.it=datos.erase(i.it);
      return iter;
  }


  iterator begin(){
      iterator i;
      i.it=datos.begin();
      return i;
  }
  iterator end(){
      iterator i;
      i.it=datos.end();
      return i;
  }
  const_iterator begin()const{
      const_iterator i;
      i.it=datos.begin();
      return i;
  }
  const_iterator end()const{
      const_iterator i;
      i.it=datos.end();
      return i;
  }
};
#endif
```

```
#include "notas.h"
#include <fstream>
#include <iostream>
using namespace std;
void Imprimir_Notas(const Notas &N){
  Notas::const_iterator it;
  for (it=N.begin(); it!=N.end();++it){
     cout<<(*it).first<<" "<<(*it).second<<endl;
  }
}
float Obtener_Media(const Notas &N){
  float suma=0;
  Notas::const_iterator it;
  for (it=N.begin(); it!=N.end();++it){
     suma+=(*it).second;
  }
  suma/=N.size();
  return suma;
}

void Modifica_Nota(Notas &N,string dni,float n){
  Notas::iterator it=N.begin();
  bool encontrado=false;
  while(it!=N.end() && !encontrado){
      if ((*it).first==dni){
        (*it).second=n;
         encontrado=true;
      }
      ++it;
  }
}



int main(int argc,char *argv[]){
 Notas N;
 if (argc!=2){
   cout<<"Dime el fichero con las notas"<<endl;
   return 0;
 }
 ifstream f (argv[1]);
 if (!f) return 0;

 f>>N; //Leemos las notas


 Imprimir_Notas(N);
 cout<<"La media de las Notas son "<<Obtener_Media(N)<<endl;

 string dni;
 float nueva_nota;
 cout<<"Dime el alumno y la nota a poner"<<endl;

 cin>>dni>>nueva_nota;
 Modifica_Nota(N,dni,nueva_nota);
 Imprimir_Notas(N);
}
```