

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 (\approx TDA Posición)
↳ 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 \Rightarrow 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.end(); ++it)
        cout << *it;
}
```

Transparencia 2

Ejercicio: crear una clase Notas en la que se almacena ~~en~~ 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.

```

1.  class Lista{
2.      private:
3.          int *datos;
4.          int n; //elementos almacenados
5.          int reservados;
6.      public:
7.          .....
8.          //DOS ITERADORES
9.          class const_iterator;
10.         class iterator{
11.             private :
12.                 int * it;
13.             public:
14.                 iterator(){}
15.                 bool operator==(const iterator &i)const{
16.                     return i.it==it;
17.                 }
18.                 bool operator!=(const iterator &i)const{
19.                     return i.it!=it;
20.                 }
21.                 int & operator *(){
22.                     return *it;
23.                 }
24.                 iterator & operator++(){
25.                     ++it;
26.                     return *this;
27.                 }
28.                 iterator & operator--(){
29.                     --it;
30.                     return *this;
31.                 }
32.             friend class Lista;
33.             friend class const_iterator;
34.         }; //fin iterator

35.         class const_iterator{
36.             private:
37.                 const int *it;
38.             public:
39.                 const_iterator(){}
40.                 const_iterator(const iterator &i):it(i.it){}
41.                 bool operator==(const const_iterator
42.                     &i)const{
43.                     return i.it==it;
44.                 }
45.                 bool operator!=(const const_iterator &i)const{
46.                     return i.it!=it;
47.                 }
48.                 const int & operator *()const{
49.                     return *it;
50.                 }
51.                 const_iterator & operator++(){
52.                     ++it;
53.                     return *this;
54.                 }
55.                 const_iterator & operator--(){
56.                     --it;
57.                     return *this;
58.                 }
59.             friend class Lista;
60.         }; //fin const_iterator
61.         iterator begin(){
62.             iterator i;
63.             i.it= &(datos[0]);
64.             return i;
65.         }
66.         iterator end(){
67.             iterator i;
68.             i.it = &(datos[n]);
69.             return i;
70.         }

71.         iterator begin(){
72.             iterator i;
73.             i.it= &(datos[0]);
74.             return i;
75.         }
76.         iterator end(){
77.             iterator i;
78.             i.it = &(datos[n]);
79.             return i;
80.         }
81.         const_iterator begin() const{
82.             const_iterator i;
83.             i.it= &(datos[0]);
84.             return i;
85.         }
86.         const_iterator end()const {
87.             const_iterator i;
88.             i.it = &(datos[n]);
89.             return i;
90.         }
91.     };

```

```

31. #ifndef __NOTAS_H
32. #define __NOTAS_H
33. 1. #include <vector>
34. 2. #include <string>
35. 3. #include <iostream>
36. 4. using namespace std;
37. 5.
38. 6. class Notas{
39. 7.     private:
40. 8.         vector<pair<string,float>> datos;
41. 9.     public:
42. 10.         Notas(){}
43. 11.         pair<string,float> & operator[](int i){ return
44.             datos[i];}
45. 12.         const pair<string,float> & operator[](int i)const
46.             { return datos[i];}
47. 13.         int size()const{ return datos.size();}
48. 14.         friend istream & operator>>(istream &is, Notas
49.             &N);
50. 15.         class const_iterator; //declaracion adelantada
51. 16.
52. //Clases Iteradores
53. 17. class iterator{
54. 18.     private:
55. 19.         vector<pair<string,float>>::iterator it;
56. 20.     public:
57. 21.         iterator(){}
58. 22.         bool operator==(const iterator &i)const{
59.             return i.it==it;
60. 23.         }
61. 24.         bool operator!=(const iterator &i)const{
62.             return i.it!=it;
63. 25.         }
64. 26.         pair<string,float> & operator*(O){
65.             return *it;
66. 27.         }
67. 28.         pair<string,float> & operator*(O){
68.             return *it;
69.         }
70.     };
71. };
72.
73. 31. iterator & operator++(O){
74. 32.     ++it;
75. 33.     return *this;
76. 34. }
77. 35. iterator & operator--(O){
78. 36.     --it;
79. 37.     return *this;
80. 38. }
81. 39. friend class Notas;
82. 40. friend class const_iterator;
83. 41. }; //fin iterator
84. 42.
85. 43. class const_iterator{
86. 44.     private:
87. 45.         vector<pair<string,float>>::const_iterator it;
88. 46.     public:
89. 47.         const_iterator(){}
90. 48.         const_iterator(const iterator &i):it(i.it){}
91. 49.         bool operator==(const const_iterator &i)const{
92.             return i.it==it;
93. 50.         }
94. 51.         bool operator!=(const const_iterator &i)const{
95.             return i.it!=it;
96. 52.         }
97. 53.         const pair<string,float> & operator*(O)const{
98.             return *it;
99. 54.         }
100. 55.         const_iterator & operator++(O){
101.             ++it;
102.             return *this;
103. 56.         }
104. 57.         const_iterator & operator--(O){
105.             --it;
106.             return *this;
107. 58.         }
108. 59.         friend class Notas;
109. 60.         }; //fin const_iterator
110. 61.
111. 62. iterator Insertar(iterator i,const
112.     string &dni,float nota){
113. 63.     pair<string,float>
114. 64.     a(dni,nota);
115. 65.     iterator iter;
116. 66.     iter.it=datos.insert(i.it,a);
117. 67.     return iter;
118. 68. }
119.
120. 69. iterator Borrar(iterator i){
121. 70.     iterator iter;
122. 71.     iter.it=datos.erase(i.it);
123. 72.     return iter;
124. 73. }
125. 74. iterator Borrar(iterator i){
126. 75.     iterator iter;
127. 76.     iter.it=datos.erase(i.it);
128. 77.     return iter;
129. 78. }
130. 79.
131. 80. iterator begin(){
132. 81.     iterator i;
133. 82.     i.it=datos.begin();
134. 83.     return i;
135. 84. }
136. 85. iterator end(){
137. 86.     iterator i;
138. 87.     i.it=datos.end();
139. 88.     return i;
140. 89. }
141. 90. const_iterator begin()const{
142. 91.     const_iterator i;
143. 92.     i.it=datos.begin();
144. 93.     return i;
145. 94. }
146. 95. const_iterator end()const{
147. 96.     const_iterator i;
148. 97.     i.it=datos.end();
149. 98.     return i;
150. 99. }
151. 100. #endif

```

nov 16, 15 17:53	notas.h	Page 1/3
<pre>#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:</pre>		
nov 16, 15 17:53	notas.h	Page 2/3
<pre>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</pre>		

```
#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);
}
```