Relacion 3 - T.D.A Lineales

Alumno: Alberto Llamas González

2º Grado Ingeniería Informática

Cuestiones

1.- ¿Es lógico tener un iterador sobre la clase Pila?

No es lógico, ya que por definición, una pila contiene una secuencia de elementos en la que está optimizada para realizar inserciones, consultas y borrados por sólo uno de los extremos (tope) siguiendo la política LIFO. Por tanto, no nos serviría de mucho tener un iterador en la clase Pila ya que únicamente apuntaría al tope y no sería útil para acceder a los elementos de nuestra clase. Aunque es cierto que para acceder al tope podríamos usar perfectamente un iterador, no sería útil tener una *clase iterator/const_iterator* ya que no lo usaríamos para acceder a los demás elementos sino única y exclusivamente al tope.

2.- ¿Es lógico mantener un iterador sobre la clase Cola?

Con las Colas ocurre algo parecido a lo que ocurre con las pilas como hemos visto en el **Ejercicio 1**. Si utilizamos iteradores sobre la clase cola, deberían de estar iniciados todo el rato ó al frente o a la última posición ya que en las *Colas* a diferencia de las *Pilas* están diseñadas para realizar consultas y borrados en el primer elemento (frente) e insertar elementos por el final (última) siguiendo una política FIFO. Por ello, ocurre lo mismo que con la clase Pila, podríamos utilizar iteradores para acceder al frente y a la última, pero no nos serviría de mucho tener una *class iterator/const_iterator*.

3.-Enumerar la diferencias entre un iterador y un const_iterator sobre un contenedor.

Normalmente se debe implementar también un const_iterator para que, al tratar con los contenedores, estemos seguros de que se mantienen los elementos de dicho contenedor. Si no usamos tenemos implementados un const_iterator, usando un iterator, podríamos usar el operador * para cambiar algún dato del contenedor. Además, como los datos de tipo *const*, un const_iterator apunta a un elemento const por lo que, aunque se pueda modificar dónde apunta el iterador, no se puede modificar el elemento al que apunta.

Otra diferencia entre ambas clases, son las funciones begin() y end() (recordemos que se deben implementar en la clase contenedora). Para ver más fácilmente la diferencia, veamos el código:

```
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.cbegin();
   return i;
}

const_iterator end() const{
   const_iterator i;
   i.it = datos.cend();
   return i;
}
```

Como podemos ver en ambas funciones, para los const_iterator tenemos los métodos cbegin() y cend() que nos devuelven const_iterators

4.-¿Por qué la clase contendora tiene que ser amiga de la clase iterator?

Porque si la clase contenedora no es amiga de la clase iterator, no podría acceder a los datos privados de la clase iterator, es decir, no podría acceder al iterator declarador como miembro de la class iterator y nos diría el compilador que no sería accesible a menos que ambas clases sean amigas. Tenemos que añadir que sea amiga la clase contenedora para que por ejemplo, el begin y el end puedan ser implementados por lo que tendrán que acceder al primer y último elemento mediante el iterator it, por tanto deben de ser ambas clases amigas. Aunque iterator sea una *nested class*, no va a poder acceder a la clase que la contiene.

5.-Suponer que tenemos la clase vector dinámico de enteros sobre la que hemos definido un iterador. Se puede definir otro iterador (iterador_par) que itere solamente sobre los elementos pares. Si es afirmativo como se representa y como se implementan las funciones miembro. (Resuelto bajo ejercicio 6)

6.- Según la cuestión 5 como se implementaría las funciones begin y end para iterator_par.

Solución ambos ejercicios:

```
#include
#include

using namespace std;

class VectorDinamico{
private:
   vector v;
public:
   class iterator_par {
   private:
      vector::iterator it;
      vector::iterator final;

   public:
      iterator_par(){}
      bool operator!=(const iterator_par &ITpar){
         return ITpar.it != this->it;
      }
      bool operator==(const iterator_par &ITpar){
         return ITpar.it == this->it;
      }
      int& operator*(){
         return *it;
      }
      iterator_par& operator++(){
         while (true){
            it++;
            if (esPar(*it)){
               return *this;
```

```
            }
            else if (it == final){
                return *this;
            }
        }
    }
    bool esPar(int entero){
        return entero % 2 == 0;
    }

    friend class VectorDinamico;
};

void aniade(int entero){
    v.push_back(entero);
}

iterator_par begin(){
    iterator_par i;
    i.it = v.begin();
    i.final = v.end();
    return i;
}

iterator_par end(){
    iterator_par i;
    i.it = v.end();
    i.final = v.end();
    return i;
}
};
```

```
int main(){
    VectorDinamico x;
    x.aniade(2);
    x.aniade(3);
    x.aniade(4);
    x.aniade(8);
    x.aniade(105);
    x.aniade(391084);
    x.aniade(1012015);

    for (auto it = x.begin(); it != x.end(); ++it){
        cout << *it << endl;
    }
}
```