Estructuras de Datos y algoritmos 1

Solución - Práctico 4

Temas: TAD Lista, TAD Pila y TAD Cola

4) Implemente una función

```
template <class T>
bool Iguales(Lista<T> *l1,Lista<T> * l2);
```

que retorna `true` si las listas `l1` y `l2` son iguales, `false` en otro caso.

```
template <class T>
bool Iguales(Lista<T> *l1,Lista<T> * l2){
      int largo1 = l1->Largo();
      int largo2 = l2->Largo();
      bool ret = false;

      if (largo1 == largo2){
            for (int i = 1; i <= largo1 &&
                  l1->ElementoEn(i) == l2->ElementoEn(i); i++);
            if(i>largo1)
                  ret = true;
      }
      return ret;
}
```

TAD Pila

6) Especifique el TAD Pila de elementos genéricos.
7) Construya una implementación acotada eficiente.
8) Construya una implementación dinámica eficiente.

```
/*************************************/
/************ "ppal.cpp" *************/
/*************************************/

#include "Articulo.h"
#include "PilaDinamicaImp.cpp"

void pruebaPila();

void main(){
      pruebaPila();
}
void pruebaPila(){
```

```
	Pila<Articulo> *p = new PilaDinamicaImp<Articulo>();
	ArticuloPtr art;
	int i;

	for (i = 1; i < 10; i++){
		art = new Articulo(i, (float)i*100);
		p->push (art);
	}
	cout<<"-----------------------------------------\n";
	art = p->buscar(5);
	if (art!= NULL)
		cout<<art->getNro()<<" - "<<art->getPrecio() <<"\n";
	cout<<"-----------------------------------------\n";
	while (!p->isEmpty()){
		art = p->top();
		p->pop();
		cout<<art->getNro()<<" - "<<art->getPrecio() <<"\n";
	}
	delete p;
}

/*************************************/
/************** "Pila.h" *************/
/*************************************/

#ifndef PILA_H
#define PILA_H

template < class Etype >
class Pila{
public:
/********** CONSTRUCTORAS ***********/
	virtual void empty()=0;
	//Crea una pila vacía

	virtual void push( Etype* x) = 0;
	//post: agrega el elemento x a la pila, en caso de que
	//la misma tenga más de n elementos se elimina el ultimo de
	// la misma

/********** SELECTORAS **************/
	virtual void pop() = 0;
	//pre: la pila debe ser no vacia
	//post: elimina el tope de la pila

	virtual Etype* top() = 0;
	//pre: la pila debe ser no vacia
	//post: retorna el tope de la pila

/********** PREDICADO ***************/
	virtual bool isEmpty() = 0;
	//post: retorna true si la pila es vacía,
	// false en otro caso
```

```
	virtual Etype* buscar(int) = 0;
};

#endif

/*************************************/
/***** "PilaDinamicaImp.cpp" *********/
/*************************************/

#ifndef PILADINAMICA_IMP
#define PILADINAMICA_IMP

#include <iostream.h>
#include <assert.h>
#include "Pila.h"

template < class Etype >
class PilaDinamicaImp : public Pila < Etype > {
/************************************/
	private:
		// Estructura que se utiliza para cada elemento de la pila
		struct Nodo{
			Etype* elemento;
			Nodo* siguiente;
		};
		typedef Nodo* PtrNodo;

		// Atributos de la clase.-
		//Puntero al ppio de la lista
		PtrNodo ppio;

	void borrarPila(){
			PtrNodo aux;
			while (ppio != NULL){
				aux = ppio;
				delete ppio->elemento;
				ppio = ppio -> siguiente;
				delete aux;
			}
		}

/************************************/
	public:
		// CONSTRUCTOR Y DESTRUCTOR //
		PilaDinamicaImp(){
			ppio = NULL;
		}

		~PilaDinamicaImp(){
			borrarPila();
		}

/********** OPERACIONES ************/
/********** CONSTRUCTORAS **********/
```

```
            void empty(){
                  borrarPila();
                  ppio = NULL;
            }

            void push(Etype *e){
                  PtrNodo nuevo;
                  nuevo = new Nodo;
                  nuevo->elemento = e;
                  nuevo->siguiente = ppio;
                  ppio = nuevo;
            }
/********** PREDICADO **************/
            bool isEmpty(){
                  return ppio == NULL;
            }

            Etype* buscar(int clave){
                  PtrNodo aux = ppio;

                  while (aux != NULL){
                        if (*(aux->elemento) == clave)
                              return aux->elemento;
                        aux = aux->siguiente;
                  }
                  return NULL;
            }
/********** SELECTORAS *************/
            Etype* top(){
                  assert(ppio!=NULL);
                  return ppio->elemento;
            }

            void pop(){
                  assert(ppio!=NULL);
                  PtrNodo borrar;
                  borrar = ppio;
                  ppio = ppio->siguiente;
                  delete borrar;
            }
};

#endif

/*************************************/
/*********** "Articulo.h" ************/
/*************************************/
#include <iostream.h>

#ifndef ARTICULO_H
#define ARTICULO_H

class Articulo{
private:
```

```
      int nro;
      float precio;

public:
/**** CONSTRUCTOR Y DESTRUCTOR ******/
      Articulo();

      Articulo(int, float);

      ~Articulo();

/********** SELECTORAS **************/
      int getNro();

      float getPrecio();

/********** PREDICADO ***************/
      void setNro(int);

      void setPrecio(float);

      void print();

      /*friend bool operator == (Articulo a, Articulo b)
      { return a.getNro () == b.getNro ();}*/

      friend bool operator == ( Articulo a, int b)
      { return a.getNro () == b;}

      friend bool operator <= (Articulo a, Articulo b)
      { return a.getNro () <= b.getNro ();}

      friend bool operator >= (Articulo a, Articulo b)
      { return a.getNro () >= b.getNro ();}

      friend bool operator < (Articulo a, Articulo b)
      { return a.getNro () < b.getNro ();}

      friend bool operator > (Articulo a, Articulo b)
      { return a.getNro () > b.getNro ();}

      friend ostream& operator << (ostream& c, Articulo a){
            return (c << a.getNro () <<" - " << a.getPrecio ());
            }


};

typedef Articulo *ArticuloPtr;

#endif

/************************************/
/**** "Articulo.cpp" *****/
```

```
/************************************/

#include "Articulo.h"

Articulo::Articulo(){
      nro = 0;
      precio = 0;
}

Articulo::Articulo(int n, float f){
      nro = n;
      precio = f;
}

Articulo::~Articulo(){
}

int Articulo::getNro(){
      return nro;
}

float Articulo::getPrecio(){
      return precio;
}

void Articulo::setNro(int n){
      nro = n;
}

void Articulo::setPrecio(float f){
      precio = f;
}

void Articulo::print(){
      cout<<nro<<" "<<precio<<"\n";
}
```

Problemas de Aplicación

13) Una doble cola es una estructura de datos consistente en una lista de elementos sobre la cual son posibles las siguientes operaciones:

   a) encolarPpio(x, d) – Inserta un elemento x en el extremo frontal de la doble cola d
   b) decolarPpio(d) – Elimina y devuelve el elemento que esta al frente de d
   c) encolarFin (x,d) - Inserta un elemento x en el extremo posterior de la doble cola d
   d) decolarFin(d) - Elimina y devuelve el elemento que esta al fondo de d

   Escriba las rutinas necesarias para implementar una doble cola de tal forma que tomen un tiempo O(1) por operación.

```
/************************************/
/************ "ppal.cpp" *************/
```

```
/************************************/
#include "Articulo.h"
#include "DequeDinamicoImp.cpp"

void pruebaDeque1(Deque<Articulo>* );

void main(){
      Deque<Articulo>* d = new DequeDinamicoImp<Articulo>();
      pruebaDeque1(d);
}

void pruebaDeque1(Deque<Articulo>* d){
      ArticuloPtr art;
      int i;

      for (i = 1; i < 10; i++){
            art = new Articulo(i, (float)i*100);
            d->encolarFin (art);
            d->encolarPpio (art);
      }

      cout<<"----------------------------------------\n";
      while (!d->isEmpty()){
            art = d->topPpio();
            d->decolarPpio();
            cout<<*art<<"\n";
            if(!d->isEmpty ()){
                  art = d->topFin();
                  d->decolarFin();
                  cout<<*art<<"\n";
            }
      }

      delete d;
}

/************************************/
/************** "Deque.h" *************/
/************************************/

#ifndef DEQUE_H
#define DEQUE_H

template < class Etype >
class Deque{
public:
/********** CONSTRUCTORAS ***********/
      virtual void empty()=0;
      //Crea una pila vacía

      virtual void encolarPpio( Etype* e) = 0;
      //post: agrega el elemento x al ppio de la doble cola

      virtual void encolarFin( Etype* e) = 0;
```

```
      //post: agrega el elemento x al final de la doble cola

/********** SELECTORAS **************/
      virtual void decolarPpio() = 0;
      //pre: la doble cola debe ser no vacia
      //post: elimina el tope de la doble cola

      virtual void decolarFin() = 0;
      //pre: la doble cola debe ser no vacia
      //post: elimina el ultimo de la doble cola

      virtual Etype* topPpio() = 0;
      //pre: la doble cola debe ser no vacia
      //post: retorna el tope de la doble cola

      virtual Etype* topFin() = 0;
      //pre: la doble cola debe ser no vacia
      //post: retorna el ultimo de la doble cola

/********** PREDICADO ***************/
      virtual bool isEmpty() = 0;
      //post: retorna true si la doble cola es vacía,
      // false en otro caso

};

#endif



/************************************/
/**** "DequeDinamicoImp.h" ***/
/************************************/

#ifndef DEQUEDINAMICOIMP_H
#define DEQUEDINAMICOIMP_H

#include <iostream.h>
#include <assert.h>
#include "Deque.h"

template < class Etype >
class DequeDinamicoImp : public Deque < Etype > {

/************************************/
      private:
            // Estructura que se utiliza para cada elemento de la pila
            struct Nodo{
                  Etype* elemento;
                  Nodo* anterior;
                  Nodo* siguiente;
            };
            typedef Nodo* PtrNodo;
```

```
            // Atributos de la clase.-
            //Puntero al ppio y al final de la lista
            PtrNodo ppio, fin;

            void borrarDeque();
/************************************/
      public:
            // CONSTRUCTOR Y DESTRUCTOR //
            DequeDinamicoImp();
            ~DequeDinamicoImp();

/********** OPERACIONES *************/
/********** CONSTRUCTORAS ***********/
            void empty();
            void encolarPpio(Etype* e);
            void encolarFin(Etype* e);

/********** PREDICADO **************/
            bool isEmpty();

/********** SELECTORAS *************/
            Etype* topPpio();
            Etype* topFin();
            void decolarPpio();
            void decolarFin();
};

#endif


/************************************/
/**** "DequeDinamicoImp.cpp" ***/
/************************************/

#ifndef DEQUEDINAMICOIMP_CPP
#define DEQUEDINAMICOIMP_CPP

#include "DequeDinamicoImp.h"


/****** OPERACIONES PRIVATES**********/
/************************************/
template < class Etype >
void DequeDinamicoImp<Etype>::borrarDeque(){
                  PtrNodo aux;
                  while (ppio != NULL){
                        aux = ppio;
                        ppio = ppio -> siguiente;
                        delete aux;
                  }
            }

/***** CONSTRUCTOR Y DESTRUCTOR ******/
/************************************/
```

```
template < class Etype >
DequeDinamicoImp<Etype>:: DequeDinamicoImp(){
                ppio = fin = NULL;
            }

template < class Etype >
DequeDinamicoImp<Etype>::~DequeDinamicoImp(){
                borrarDeque();
            }

/********** OPERACIONES *************/
/********** CONSTRUCTORAS ***********/
template < class Etype >
void DequeDinamicoImp<Etype>::empty(){
                borrarDeque();
                ppio = fin = NULL;
            }

template < class Etype >
void DequeDinamicoImp<Etype>::encolarPpio(Etype* e){
                PtrNodo nuevo;
                nuevo = new Nodo;
                nuevo->elemento = e;
                nuevo->siguiente = ppio;
                nuevo->anterior = NULL;
                if (ppio != NULL)
                      ppio->anterior = nuevo;
                ppio = nuevo;
                if (fin == NULL)
                      fin = ppio;
            }

template < class Etype >
void DequeDinamicoImp<Etype>::encolarFin(Etype* e){
                PtrNodo nuevo;
                nuevo = new Nodo;
                nuevo->elemento = e;
                nuevo->siguiente = NULL;
                nuevo->anterior = fin;
                if (fin != NULL)
                      fin->siguiente = nuevo;
                fin = nuevo;
                if (ppio == NULL)
                      ppio = fin;

            }

/********** PREDICADO **************/
template < class Etype >
bool DequeDinamicoImp<Etype>::isEmpty(){
                return ppio == NULL;
            }
/********** SELECTORAS *************/
```

```
template < class Etype >
Etype* DequeDinamicoImp<Etype>::topPpio(){
                  assert(ppio!=NULL);
                  return ppio->elemento;
            }

template < class Etype >
Etype* DequeDinamicoImp<Etype>::topFin(){
            assert(fin!=NULL);
            return fin->elemento;
      }

template < class Etype >
void DequeDinamicoImp<Etype>::decolarPpio(){
                  assert(ppio!=NULL);
                  PtrNodo borrar;
                  borrar = ppio;
                  ppio = ppio->siguiente;
                  if(ppio!= NULL)
                        ppio->anterior =NULL;
                  else
                        ppio = fin = NULL;
                  delete borrar;
            }

template < class Etype >
void DequeDinamicoImp<Etype>::decolarFin(){
            PtrNodo borrar;
            assert(ppio != NULL);
            borrar = fin;
            if(fin->anterior != NULL){
                  fin = fin->anterior;
                  fin->siguiente = NULL;
            }else
                  ppio = fin = NULL;
            delete borrar;
            }
#endif

/*********************************************************************/
/* La clase articulo es la misma que se mostró en la solución anterior */
/*********************************************************************/
```