Facultad de Ingeniería y Ciencias Hídricas
FICH
UNL

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Ingeniería Informática

PROGRAMACIÓN ORIENTADA A OBJETOS

UNIDAD 3

Estructuras de datos dinámicas

2011

UNIDAD 3

Estructuras de datos dinámicas

Ejercicio 3.1

```
struct Nodo {
    int valor;
    Nodo *siguiente;
    Nodo(int v, Nodo *sig = NULL){
        valor = v;
        siguiente = sig;
    }
};
```

Utilizando la estructura *Nodo* mostrada en el recuadro, implemente una lista simplemente enlazada con los siguientes métodos:

- Un constructor por defecto.
- Un destructor que libere, en caso de ser necesario, la memoria reservada.
- Un método *PushFront()*, que reciba un entero y lo inserte al principio de la lista.
- Un método *PushBack()*, que reciba un entero y lo inserte al final de la lista.
- Un método *PopFront()*, que elimine el primer elemento de la lista.
- Un método *Clear()*, que elimine todos los elementos de la lista, dejándola vacía.
- Un método *Begin()* que devuelva un puntero al primer Nodo de la lista. ¿Que problemas puede traer este método? Si encuentra problemas, modifique la clase Nodo para evitarlos.
- **(OPCIONAL)** Un constructor de copia

Escriba un programa cliente que muestre un menú con opciones para agregar y quitar nodos y para ver el contenido de la lista.

Ejercicio 3.2

Reutilice la clase del ejercicio anterior para crear, utilizando herencia, una clase *Pila* que permita representar una pila de enteros. Averigüe como utilizar ocultamiento de datos para que solo las siguientes funciones estén disponibles para el usuario.

- Un constructor que no reciba parámetros e inicialize una pila vacía.
- Un método *Push()* que reciba un entero y lo apile.
- Un método *Top()* que devuelva el elemento en el tope de la pila.
- Un método *Pop()* que quite el elemento que está en el tope de la pila.

- Un método *Size()* que devuelva la cantidad de datos apilados.
- Un método *Clear()* que desapile todos los elementos.
- Un destructor que libere toda la memoria reservada.

Escriba un programa cliente que agregue 10 elementos aleatorios y luego los muestre y quite de la lista para corroborar que el orden en que se apilan y desapilan sea el correcto.

Ejercicio 3.3

Reutilice la clase del ejercicio anterior para crear, utilizando herencia, una clase Cola que permita representar una cola de enteros. Averigüe como utilizar ocultamiento de datos para que solo las siguientes funciones estén disponibles para el usuario.

- Un constructor que no reciba parámetros e inicialize una pila vacía.
- Un método *Push()* que reciba un entero y lo encole.
- Un método Front*()* que devuelva el elemento en el frente de la cola.
- Un método *Pop()* que quite el elemento que está en el frente de la cola.
- Un método *Size()* que devuelva la cantidad de datos apilados.
- Un método *Clear()* que desapile todos los elementos aplicados.
- Un destructor que libere toda la memoria reservada.

Escriba un programa cliente que agregue 10 elementos aleatorios y luego los muestre y quite de la lista para corroborar que el orden en que se muestran sea el correcto.

Ejercicio 3.4

El departamento de informática de la FICH dispone de tres impresoras: la impresora A solo imprime en blanco y negro, la impresora B imprime en color, pero con baja calidad, mientras que la impresora C se utiliza sólo para los trabajos color de alta calidad. Escriba un programa que lea un conjunto de trabajos para imprimir, cada trabajo tiene la información del struct mostrado debajo. El programa debe confeccionar las tres colas de impresión con los trabajos leídos según sus propiedades y su orden de llegada. Al final, debe mostrar cada una de ellas.

```
struct Trabajo {
    char nombre[50];
    bool color;
    bool altaCalidad;
};
```

Ejercicio 3.5

```
struct Medicion {
    int Dia, Mes, Anio;
    float Medicion;
};
```

Defina una Pila que contenga como elementos, datos del tipo Medicion mostrado arriba, que representan mediciones tomadas del caudal de un río. La clase debe poseer los mismos métodos que la Pila desarrollada en el ejercicio 4.1. Además, debe incluirse:

- Una sobrecarga del método *Push()* que reciba el dato a apilar y un entero con la cantidad de veces que el mismo debe ser apilado.
- Un sobrecarga del método *Pop()* que reciba dos parámetros: la cantidad de datos que deben desapilarse y un arreglo donde se almacenarán los datos desapilados.
- **(OPCIONAL)** Implemente un método *RobaLista()* que reciba una lista y mueva todos los elementos de la lista al final de la pila, sin crear ni destruir nodos.

Desarrolle además un programa cliente que haga uso de las funciones mencionadas arriba.

Ejercicio 3.6 (OPCIONAL)

Implemente una clase *DLista* para modelar una lista doblemente enlazada de flotantes. La lista debe poseer los mismos métodos que la lista implementada en el ejercicio 3.1, y además deben agregarse los siguientes métodos.

- Un metodo *Insert()* que reciba el valor a insertar y un puntero al nodo en la posición donde debe insertarse dicho valor.
- Un metodo *Erase()* que reciba un puntero al nodo que debe ser eliminado.

Cuestionario

- ¿Qué diferencias existen entre una pila, una cola y una lista?
- ¿Cuáles ejemplos puede mencionar como usos prácticos para una estructura de tipo pila?¿Y de una estructura de tipo lista?
- ¿Qué diferencias existen, en cuanto a la reserva de memoria, entre un vector y una lista?
- ¿Qué diferencias existen entre una lista simplemente enlazada y una lista doblemente enlazada?
- Responda según su criterio, si la inserción de un elemento dentro de una lista es computacionalmente más costosa que la inserción dentro de un vector. Justifique su respuesta.
- ¿Qué significa que una estructura sea de acceso aleatorio?¿Y acceso secuencial?
- La lista, ¿posee acceso aleatorio o secuencial?¿y el vector?
- ¿Qué ventajas tiene poseer acceso aleatorio a una estructura de datos?