4punto0
www.wuolah.com/student/4punto0

Ordinario-2018.pdf

Ordinario 2018 + SOLUCIÓN

3° Informática Industrial I

Grado en Ingeniería Electrónica Industrial y Automática

Escuela Politécnica Superior
Universidad Carlos III de Madrid

INFORMÁTICA INDUSTRIAL

Grado en Ingeniería en Electrónica Industrial y Automática
Examen Final
19 de Enero de 2018

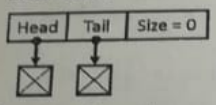
Nombre:..... Grupo.....

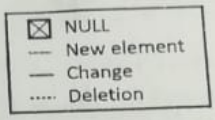
Nota: Escribir el nombre en **todas** las hojas, en cuanto las recibas.

Ejercicio 2 – 3.5 puntos

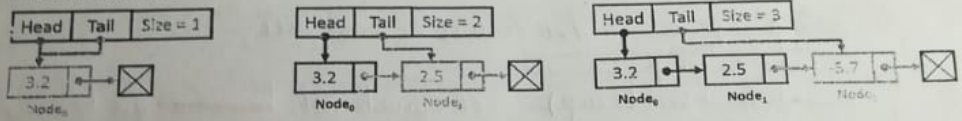
Una **lista enlazada** es una colección de datos, donde el orden lineal no viene dado el lugar físico que ocupan en la memoria. Por el contrario, cada elemento apunta al siguiente de la lista. Es una estructura de datos que consiste en un grupo de nodos que representan una secuencia. Cada nodo se compone de un campo de datos y una referencia (o enlace) al siguiente nodo de la secuencia. La lista enlazada que se va a implementar contiene tres atributos: un puntero al primer elemento de la lista (*head*), un puntero al último elemento (*tail*) y un entero que almacena el número de total de nodos de la secuencia (*size*). A continuación, se muestra el funcionamiento de las operaciones básicas sobre una lista enlazada de elementos tipo **float**:

Lista vacía:

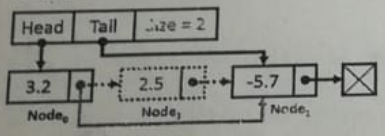




Insertar elemento:



Borrar elemento:



La clase Node es una plantilla que se emplea para almacenar los elementos de una lista enlazada.

1) Rellena los campos en el archivo de cabecera de la clase.

```
node.h
template<class N> // 1) Declaración de la plantilla
class Node        // 2)
{
private:
    N     _data; // 3) Variable para almacenar los datos
    Node* _next; // 4) Puntero al siguiente elemento Node (de la lista)
public:
    Node(N data);// 5) Constructor parametrizado
    N     get(); // 6) Función para acceder al valor del nodo
    Node* next(); // 7) Función para acceder a _next
    void set(N valor); // 8) Función para establecer _data
    void setNext(Node* n); // 9) Función para establecer _next
};
```

2) A continuación, completa el archivo *node.h* con la implementación de los métodos.

```
// 10) Implementación de la plantilla → .cpp
```

LinkedList es una plantilla usada para almacenar una lista de elementos de cualquier tipo. Para ello, hace uso de la clase ***Node***.

3) Completa el código rellenando los espacios en blanco.

linkedlist.h

```
#include <iostream>
#include "node.h"

using namespace std;

template <class N>// 11) Declaración de la plantilla
class LinkedList // 12) Clase para almacenar una lista de elementos Node
{
private:
    Node* _head; // 13) Puntero al primer elemento de la lista
    Node* _tail; // 14) Puntero al último elemento de la lista
    int _size; // Almacena el número de elementos de la lista

public:
    LinkedList() { // 15) Constructor vacio. Inicializar campos para lista vacia
        _head = NULL;
        _tail = NULL;
        _size = 0;
    }

    int size() {return _size;} // Devuelve el tamaño de la lista

    void insert(N data){ // 16) Inserta un elemento al final de la lista
        // 17) Puntero al nuevo nodo con el valor a insertar
        Node* temp = new Node(data);

        // 18) Si la lista está vacia, _head y _tail apuntan al nuevo Node
        if(_size==0){ // _tail==NULL && _head==NULL
            _tail=temp;
            _head=temp;
        }else{ // 19) Insertar el nuevo Node al final de la lista
            _tail.setNext(temp);
        }
        _tail = temp; // 20) Actualizar _tail
        _size++; // 21) Actualizar el tamaño
    }

    bool removeAt(int index){ // Borrar el elemento de la posición indicada
        // 22) Comprobar que index no sale de rango
        if(_size<=index){
            // 23) Imprimir mensaje de acuerdo con la salida de consola proporcionada
            cout<<"index out of bounds. Unable to remove element"<<endl;

            // Devolver false para indicar que la operación no pudo realizarse
            return false;
        }
        // 24) Puntero auxiliar al nodo actual. Inicializado en _head
        Node* current = _head;

        // 25) Puntero auxiliar al nodo previo. Inicializado vacío
        Node* prev = NULL;

        // 26) Mover el puntero current al nodo que queremos eliminar
        for(int i=0; i<index; i++){
            prev = current;
            current = current->next();
        }
        if(prev==NULL){ // 27) Si borramos el primer elemento
            _head = head->next(); // 28) Actualizar _head
        }else{ // 29) Conectar prev con el siguiente nodo saltándonos current
            prev->next = current->next();
        }
```

```
        if(current->next()== NULL ){ // 30) Si borramos el último elemento
            _tail = prev ; // 31) Actualizar _tail
        delete current ; // 32) Borramos el contenido del puntero current
        _size--; // 33) Actualizar el tamaño
        return true; // Devolver true para indicar que la operación se realizó
    }

    void display(){ // Muestra por pantalla los elementos de la lista
        Node* temp = _head; // 34) Puntero al Node a imprimir, empieza en _head
        cout << "List: [ ";
        for(int i=0; i<_size; i++){
            cout << temp->get(); // Imprimir valor del nodo
            temp = temp->next(); // Ir al siguiente nodo
            if(i!=(_size-1)){
                cout << ", ";
            }
        }
        cout << " ]" << endl;
    }
};
```

Por comodidad, se proporciona un fichero *main.cpp* que emplea la clase ***LinkedList*** para guardar una lista de números enteros, así como la salida por consola correspondiente.

main.cpp

```
#include <iostream>
#include "linkedlist.h"
using namespace std;

int main(){
    LinkedList<int> my_list; // Creando una lista vacia de enteros

    // Usando las funciones insert/removeAt/display
    my_list.display();
    my_list.insert(8);
    my_list.display();
    my_list.removeAt(0);
    my_list.display();
    my_list.insert(5);
    my_list.display();
    my_list.insert(6);
    my_list.insert(7);
    my_list.display();
    my_list.removeAt(1);
    my_list.display();
    my_list.removeAt(my_list.size()-1);
    my_list.display();

    // Intentando borrar un elemento no existente
    my_list.removeAt(100);
    return 0;
}
```

Console output: