Documento anónimo

Ejercicio-Celda-y-Lista-Completo.pdf

Ejercicio de Celda y Lista Completo

1º Metodología de la Programación

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

```
#include <iostream>
#include <cstdlib>

using namespace std;

struct Celda{

  int dato; //Indica el valor de la celda
  Celda *siguiente; //Indica la dirección de memoria de la celda siguiente, si no hay siguiente será NULL.

};

class Lista{

  private:

    Celda *comienzo; //Puntero que apunta al comienzo de la lista.
    Celda *fin; //Puntero que apunta al final de la lista.

  public:

    Lista(){ //Constructor por defecto de la clase Lista.

      comienzo = NULL; //Como no hay ninguna celda, comienzo y fin apuntan a NULL
      fin = NULL;

    }

    void Aniadir(int valor){ //Método que añade una celda al final de la lista.

      Celda *temporal = new Celda; //Creamos una celda.
      temporal->dato = valor; //El valor de esta celda será el introducido por el usuario.
      temporal->siguiente = NULL; //Como es la última, su puntero a siguiente será NULL

      if(comienzo == NULL){ //Si el comienzo es NULL, es decir que no hay ninguna celda, se establecen los parámetros siguientes.

        comienzo = temporal; //El comienzo apuuntará a la celda.
        fin = temporal; //El final apuntará a la celda.
        temporal = NULL; //Como es la primera celda, su celda a siguiente será NULL.

      }else{ //Si ya hay otras celdas.

        fin->siguiente = temporal; //El puntero que apunta al final, apuntará a esta nueva celda.
        fin = temporal; //Final apunta a la celda.

      }

    }

    void Mostrar(){ //Método que muestra los valores de la lista.
```

```
    Celda *temporal = new Celda; //Para no modificar la lista, creamos una celda que apunte al comienzo.
    temporal = comienzo;

    while(temporal != NULL){ //Hasta que el puntero que apunta al siguiente no sea NULL, leer los datos de la lista.

      cout << temporal->dato << " ";
      temporal = temporal->siguiente;

    }

  }

  void InsertarComienzo(int valor){ //Método que introduce un valor al comienzo de la lista.

    Celda *temporal = new Celda; //Creamos la celda.
    temporal->dato = valor; //Le añadimos el valor que el usuario nos indica.
    temporal->siguiente = comienzo; //Su puntero a siguiente, apuntará al comienzo anterior.
    comienzo = temporal; //El nuevo comienzo apunta a nuestra celda.

  }

  void InsertarPosicion(int valor, int pos){ //Método que inserta una celda dado una posición.

    Celda *previo = new Celda; //Creamos la celda previa a donde vamos a insertar la nueva.
    Celda *actual = new Celda; //Creamo la celda actual donde se va a insertar la nueva, es decir, la que estará después de introducir la nueva.
    Celda *temporal = new Celda; //Creamos la celda nueva.

    actual = comienzo; //Actual apunta al comienzo.

    for(int i = 1; i < pos; i++){ //A previo le vamos añadiendo todos los valores de la LIsta.

      previo = actual;
      actual = actual->siguiente;

    }

    temporal->dato = valor; //Le añadimos a nuestra celda el valor introducido por el usuario.
    previo->siguiente = temporal; //El puntero de la celda previa va a apuntar a nuestra nueva celda.
    temporal->siguiente = actual; //El puntero de la celda nueva va a apuntar a nuestra celda actual, es decir la que se posiciona delante.

  }

  void EliminarComienzo(){ //Método que elimina el elemento del comienzo.

    Celda *temporal = new Celda; //Creamos una nueva celda.
    temporal = comienzo; //La nueva celda apuntará al comienzo.
    comienzo = temporal->siguiente; //La posición a donde apunta será la siguiente.
```

```
    delete temporal; //Eliminamos la celda creada.

  }

  void EliminarFinal(){ //Método que elimina el elemento del final.

    Celda *actual = new Celda; //Creamos una celda actual.
    Celda *previo = new Celda; //Creamos una celda previa, que apunte a la celda previa a la que deseamos eliminar.
    actual = comienzo; //Actual apunta al comienzo.

    while(actual->siguiente != NULL){ //Hasta que actual llegue al final, previo será igual a actual.

      previo = actual;
      actual = actual->siguiente;

    }

    fin = previo; //Final apuntará al previo.
    previo->siguiente = NULL; //El puntero de previo apuntará a NULL, ya que será la última celda.
    delete actual; //Eliminamos actual.

  }

  void EliminarPosicion(int pos){ //Método que elimina una celda en determinada posición.

    Celda *actual = new Celda; //Creamos la celda actual.
    Celda *previo = new Celda; //Creamos la celda previa.
    actual = comienzo; //Actual apuntará al comienzo.

    for(int i = 1; i < pos; i++){ //Vamos copiando la lista en previo hasta la posición indicada por el usuario.

      previo = actual;
      actual = actual->siguiente;

    }

    previo->siguiente = actual->siguiente; // El puntero a siguiente del previo va a apuntar a al siguiente del actual.

  }

  void OrdenarCre(){ //Método que ordena de forma creciente una lista.

    Celda *temporal = comienzo; //Creamos una celda temporal que apunta al comienzo.

    while(temporal != NULL){ //Mientras la celda no llegue al final de la lista

      Celda *i = temporal->siguiente; //Creamos otra celda que apunta a la celda siguiente de temporal.
```

```
    while(i != NULL){ //Hasta que esta última celda llegue al final.

      if(temporal->dato > i->dato){ //Si el dato de la celda anterior es mayor que la celda actual, se intercambian.

        int aux = i->dato;
        i->dato = temporal->dato;
        temporal->dato = aux;

      }

      i = i->siguiente; //Vamos aumentando a donde apunta.

    }

    temporal = temporal->siguiente; //Lo mismo para la celda anterior.

  }

}

void OrdenarDecre(){ //Método que ordena de forma decreciente la lista.

  Celda *temporal = comienzo;

  while(temporal != NULL){

    Celda *i = temporal->siguiente;

    while(i != NULL){

      if(temporal->dato < i->dato){

        int aux = i->dato;
        i->dato = temporal->dato;
        temporal->dato = aux;

      }

      i = i->siguiente;

    }

    temporal = temporal->siguiente;

  }

}

void EliminaValor(int valor){ //Método que elimina un elemnto indicado por el usuario.
```

```
    Celda *temporal = new Celda; //Creamos una celda temporal que apuntará al comienzo
    temporal = comienzo;
    int contador = 1;
    bool encontrado = false;

    while(temporal != NULL && encontrado == false){ //Recorremos la lista hasta que
encontremos el valor indicado o hasta que se acabe.

      if(temporal->dato == valor){ //Si encontramos el valor, encontrado es cierto, eliminamos la
celda de esa posición.

        encontrado = true;
        EliminarPosicion(contador);

      }

      contador ++;
      temporal = temporal->siguiente; //Vamos recorriendo la lista.

    }

  }

};

int main(){

  int numero = 0;
  int val = 0;
  int posicion = 0;

  Lista l1;

  while(numero != -1){

    cout << endl << "Pulse 1 para agregar un valor al final. ";
    cout << endl << "Pulse 2 para agregar un valor al comienzo. ";
    cout << endl << "Pulse 3 para agregar un valor en cierta posicion. ";
    cout << endl << "Pulse 4 para eliminar un valor del comienzo. ";
    cout << endl << "Pulse 5 para eliminar un valor del final. ";
    cout << endl << "Pulse 6 para eliminar un valor de cierta posicion. ";
    cout << endl << "Pulse 7 para mostrar su lista. ";
    cout << endl << "Pulse 8 para ordenar su lista de menor a mayor. ";
    cout << endl << "Pulse 9 para ordenar su lista de mayor a menor. ";
    cout << endl << "Pulse 10 para eliminar cierto elemento. ";
    cout << endl << "Pulse -1 para terminar. ";

    cout << endl << "Introduzca una cifra: ";
    cin >> numero;

    cout << "\033[2J\033[1;1H";
```

```
if(numero == 1){

  cout << endl << "Introduzca su valor para ser introducido al final: ";
  cin >> val;
  l1.Aniadir(val);

}else if(numero == 2){

  cout << endl << "Introduzca su valor para ser introducido al comienzo: ";
  cin >> val;
  l1.InsertarComienzo(val);

}else if(numero == 3){

  cout << endl << "Introduzca su valor y su posicion donde desea insertarlo: ";
  cin >> val >> posicion;
  l1.InsertarPosicion(val, posicion);

}else if(numero == 4){

  cout << endl << "Valor eliminado del principio.";
  l1.EliminarComienzo();

}else if(numero == 5){

  cout << endl << "Valor eliminado del final.";
  l1.EliminarFinal();

}else if(numero == 6){

  cout << endl << "Introduzca la posicion que desea eliminar: ";
  cin >> posicion;
  l1.EliminarPosicion(posicion);

}else if(numero == 7){

  cout << endl << "Su lista es la siguiente: ";
  l1.Mostrar();

}else if(numero == 8){

  l1.OrdenarCre();

}else if(numero == 9){

  l1.OrdenarDecre();

}else if(numero == 10){

  cout << endl << "Introduzca la cifra que desea eliminar: ";
  cin >> val;
  l1.EliminaValor(val);
```

```
    }
  }
}
```