UNSA

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

“UNIVERSIDAD NACIONAL DE SAN AGUSTÍN”

FACULTAD DE INGENIERÍA,PRODUCCIÓN Y SERVICIOS
ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

CURSO:

Ciencias de la Computación - Grupo “B”

DOCENTE:

Enzo Edir Velásquez Lobatón

ALUMNO:

Fabricio Huaquisto Quispe

REPOSITORIO:

https://github.com/fhuaquisto21/EPCC-CCII

Arequipa - Perú

2022

1. **list.h**

```
/**
 * 1. Defina una lista enlazada que permita insertar elementos al final de todos los
 * elementos que ya se hayan ingresado. Por el momento no es necesario preservar un
 * orden, simplemente los elementos nuevos deben de ingresar como el último elemento.
 * */

#include "node.cpp"
class List {
  private:
    Node* head;
    int length;
    void orderAscending(int*, int);
    void orderDescending(int*, int);
  public:
    List();
    List(int);
    ~List();
    Node* addLastNode(int);

    /**
     * 2. Con la implementación de la lista enlazada anterior, desarrolle una función que
     * permita ingresar los elementos al inicio de todos los demás elementos. Tendrá que
     * modificar el comportamiento del puntero que tiene referencia al primer elemento para
     * que sea redireccionado al nuevo elemento por ingresar.
     * */
    Node* addFirstNode(int);

    /**
     * 3. Desarrolle una función que permita ingresar elementos en el medio de dos elementos
     * de la lista enlazada, como se muestra en la siguiente imagen. Solicite que se ingrese
     * una posición válida dentro de la lista y permita que el valor ingresado se pueda anexar
     * en esa posición.
     * */
    Node* addIndexNode(int, int);

    /**
     * 4. Elabore una función que permita eliminar el último elemento de una lista enlazada.
```

```
     * (Evite copiar los elementos en una nueva lista para completar la
eliminación del
     * elemento)
     * */
    void deleteLastNode();

    /**
     * 5. Desarrolle una función que permita eliminar el primer elemento de
una lista sin perder
     * referencia de los demás elementos que ya se encuentran almacenados
en la estructura.
     * (Evite copiar los elementos en una nueva lista para completar la
eliminación de los
     * elementos)
     * */
    void deleteFirstNode();

    /**
     * 6. Dado una posición válida dentro de la lista, permita al usuario
eliminar un elemento
     * de cualquier posición sin perder referencia de los demás elementos.
     * */
    void deleteIndexNode(int);

    /**
     * 7. Desarrolle un algoritmo de ordenamiento que permita ordenar los
elementos de forma
     * ascendente y descendente de acuerdo a la elección del usuario. Se
debe poder simular
     * el ingreso de 10 mil elementos de forma aleatoria y ordenarlos en el
menor tiempo
     * posible ( < 2 seg).
     * */
    void ascendingOrder();
    void descendingOrder();
    void printList();
};
```

2. **list.cpp**

```
/**
 * 1. Defina una lista enlazada que permita insertar elementos al final de todos los
 * elementos que ya se hayan ingresado. Por el momento no es necesario preservar un
 * orden, simplemente los elementos nuevos deben de ingresar como el último elemento.
 */
#include <iostream>
#include "list.h"

using namespace std;

List::~List() {}

List::List() {
  this->head = nullptr;
  this->length = 0;
}

List::List(int _value) {
  this->head = new Node(_value);
  this->length = 1;
}

Node* List::addLastNode(int _value) {
  Node* currentNode = this->head;
  Node* newNode = new Node(_value);
  if (currentNode == nullptr) {
    this->head = newNode;
    ++this->length;
    return this->head;
  } else {
    while (currentNode->getNext() != nullptr) {
      currentNode = currentNode->getNext();
    }
    currentNode->setNext(newNode);
  }
  ++this->length;
  return currentNode->getNext();
}

/**
 * 2. Con la implementación de la lista enlazada anterior, desarrolle una función que
 * permita ingresar los elementos al inicio de todos los demás elementos. Tendrá que
```

```
 * modificar el comportamiento del puntero que tiene referencia al primer elemento para
 * que sea redireccionado al nuevo elemento por ingresar.
 * */
Node* List::addFirstNode(int _value) {
  Node* newNode = new Node(_value);
  newNode->setNext(this->head);
  this->head = newNode;
  ++this->length;
  return this->head;
}

/**
 * 3. Desarrolle una función que permita ingresar elementos en el medio de dos elementos
 * de la lista enlazada, como se muestra en la siguiente imagen. Solicite que se ingrese
 * una posición válida dentro de la lista y permita que el valor ingresado se pueda anexar
 * en esa posición.
 * */
Node* List::addIndexNode(int _value, int _index) {
  Node* currentNode = this->head;
  Node* prevNode = nullptr;
  Node* newNode = new Node(_value);
  if (_index >= this->length) {
    cout << "ERROR: El índice no existe.";
    return nullptr;
  }
  if (_index == 0) {
    return currentNode;
  }
  for (int i = 1; i <= _index; ++i) {
    prevNode = currentNode;
    currentNode = currentNode->getNext();
  }
  newNode->setNext(currentNode);
  prevNode->setNext(newNode);
  ++this->length;
  return prevNode->getNext();
}


/**
 * 4. Elabore una función que permita eliminar el último elemento de una lista enlazada.
 * (Evite copiar los elementos en una nueva lista para completar la eliminación del
```

```
 * elemento)
 * */
void List::deleteLastNode() {
   Node* currentNode = this->head;
   Node* prevNode = nullptr;
   while (currentNode->getNext() != nullptr) {
      prevNode = currentNode;
      currentNode = currentNode->getNext();
   }
   delete currentNode;
   prevNode->setNext(nullptr);
   --this->length;
}

/**
 * 5. Desarrolle una función que permita eliminar el primer elemento de una
lista sin perder
 * referencia de los demás elementos que ya se encuentran almacenados en
la estructura.
 * (Evite copiar los elementos en una nueva lista para completar la
eliminación de los
 * elementos)
 * */
void List::deleteFirstNode() {
   Node* currentNode = this->head;
   Node* nextNode = currentNode->getNext();
   delete currentNode;
   this->head = nextNode;
   --this->length;
}

/**
 * 6. Dado una posición válida dentro de la lista, permita al usuario eliminar
un elemento
 * de cualquier posición sin perder referencia de los demás elementos.
 * */
void List::deleteIndexNode(int _index) {
   Node* currentNode = this->head;
   Node* prevNode = nullptr;
   if (_index >= this->length) {
      cout << "ERROR: El índice no existe.";
   }
   if (_index == 0) {
      Node* nextNode = currentNode->getNext();
      delete currentNode;
      this->head = nextNode;
   }
   for (int i = 1; i <= _index; ++i) {
```

```
      prevNode = currentNode;
      currentNode = currentNode->getNext();
    }
    Node* nextNode = currentNode->getNext();
    delete currentNode;
    prevNode->setNext(nextNode);
    --this->length;
}

void List::ascendingOrder() {
    for (int i = 0; i < this->length; ++i) {
        Node* currentNode = this->head;
        Node* nextNode = currentNode->getNext();
        for (int j = 0; j < this->length - 1; ++j) {
            if (currentNode->getValue() > nextNode->getValue()) {
                int aux = currentNode->getValue();
                currentNode->setValue(nextNode->getValue());
                nextNode->setValue(aux);
            }
            currentNode = nextNode;
            nextNode = nextNode->getNext();
        }
    }
}

/**
 * 7. Desarrolle un algoritmo de ordenamiento que permita ordenar los
elementos de forma
 * ascendente y descendente de acuerdo a la elección del usuario. Se debe
poder simular
 * el ingreso de 10 mil elementos de forma aleatoria y ordenarlos en el menor
tiempo
 * posible ( < 2 seg).
 * */
void List::descendingOrder() {
    for (int i = 0; i < this->length; ++i) {
        Node* currentNode = this->head;
        Node* nextNode = currentNode->getNext();
        for (int j = 0; j < this->length - 1; ++j) {
            if (currentNode->getValue() < nextNode->getValue()) {
                int aux = currentNode->getValue();
                currentNode->setValue(nextNode->getValue());
                nextNode->setValue(aux);
            }
            currentNode = nextNode;
            nextNode = nextNode->getNext();
        }
    }
```

```
}

void List::printList() {
   Node* currentNode = this->head;
   if (currentNode == nullptr) {
      cout << "La lista está vacía." << endl;
   } else {
      while (currentNode->getNext() != nullptr) {
         cout << currentNode->getValue() << " -> ";
         currentNode = currentNode->getNext();
      }
      cout << currentNode->getValue() << endl;
   }
}
```

3. **node.h**

```
/**
 * 1. Defina una lista enlazada que permita insertar elementos al final de todos los
 * elementos que ya se hayan ingresado. Por el momento no es necesario preservar un
 * orden, simplemente los elementos nuevos deben de ingresar como el último elemento.
 */
class Node {
  private:
    Node* next;
    int value;
  public:
    Node();
    Node(int);
    ~Node();
    Node* getNext();
    Node* setNext(Node*);
    int getValue();
    int setValue(int);
};
```

4. **node.cpp**

```
/**
 * 1. Defina una lista enlazada que permita insertar elementos al final de todos los
 * elementos que ya se hayan ingresado. Por el momento no es necesario preservar un
 * orden, simplemente los elementos nuevos deben de ingresar como el último elemento.
 */
#include "node.h"

Node::~Node() {}

Node::Node() {
  this->value = 0;
  this->next = nullptr;
}

Node::Node(int _value) {
  this->value = _value;
  this->next = nullptr;
}

Node* Node::getNext() {
  return this->next;
}

Node* Node::setNext(Node *_next) {
  this->next = _next;
  return this->next;
}

int Node::setValue(int _value) {
  this->value = _value;
  return this->value;
}

int Node::getValue() {
  return this->value;
}
```

5. **main.cpp**

```
#include <iostream>
#include "list.cpp"
using namespace std;

void printMenu() {
  cout << "[1] Agregar nodo al final" << endl;
  cout << "[2] Agregar nodo al principio" << endl;
  cout << "[3] Agregar nodo por índice" << endl;
  cout << "[4] Eliminar el último nodo" << endl;
  cout << "[5] Eliminar el primer nodo" << endl;
  cout << "[6] Eliminar nodo por índice" << endl;
  cout << "[7] Ordenar ascendentemente la lista" << endl;
  cout << "[8] Ordenar descendentemente la lista" << endl;
  cout << "[9] Mostrat lista" << endl;
  cout << "[0] Salir" << endl;
  cout << endl << "Option: ";
}

int main() {
  List* lista = new List();
  int opt;
  int index, value;
  do {
    printMenu();
    cin >> opt;
    printf("\e[1;1H\e[2J");
    switch (opt) {
      case 0:
        break;
      case 1:
        cout << endl << "Valor del nodo: ";
        cin >> value;
        lista->addLastNode(value);
        break;
      case 2:
        cout << endl << "Valor del nodo: ";
        cin >> value;
        lista->addFirstNode(value);
        break;
      case 3:
        cout << endl << "Valor del nodo: ";
        cin >> value;
        cout << "Índice del nodo: ";
        cin >> index;
        lista->addIndexNode(value, index);
        break;
      case 4:
```

```
            lista->deleteLastNode();
            break;
        case 5:
            lista->deleteFirstNode();
            break;
        case 6:
            cout << endl << "Índice del nodo: ";
            cin >> index;
            lista->deleteIndexNode(index);
            break;
        case 7:
            lista->ascendingOrder();
            break;
        case 8:
            lista->descendingOrder();
            break;
        case 9:
            cout << endl;
            lista->printList();
            cout << endl;
            char temp;
            temp = cin.get();
            break;
      }
    } while (opt != 0);
}
```