Ingeniería en computación.

Estructura de datos I

Sección: D14

Listas estáticas

Integrantes:

González Rodríguez Santiago

Gutiérrez García Gwyneth Livier

Guzmán Rodríguez Gerardo

Hernandez Pizano Miguel Emmanuel

López Arellano Ricardo David

Martínez González Edgar Agustín

Calendario:2021A.

Profesor: Sánchez Estrada, Jairo Cain.

Guadalajara Jalisco, México

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

int main() {
    list list;

    int opcion = 0;
    int info = 0;
    int pos = 0;
    bool result = false;
    node *aux = nullptr;

    do {
        std::cout
                << "1.- Insertar\n2.- Eliminar\n3.- Primer Elemento\n4.-
                Ultimo Elemento\n5.- Busqueda\n6.- Recupera \n7.- Imprime\n0.- Salir\n";
        std::cin >> opcion;
        switch (opcion) {
            case 1: {
                std::cout << "Ingrese el numero a insertar: ";
                std::cin >> info;

                std::cout << "Ingrese la posicion en la que desea insertar: ";
                std::cin >> pos;
                aux = list.findByIndex(pos);

                list.add(info, aux);
                std::cout << "Numero agregado exitosamente\n";
                break;
            }
            case 2: {
                std::cout << "Ingrese la posicion que desea eliminar: ";
                std::cin >> pos;
                aux = list.findByIndex(pos);
                list.remove(aux);
                std::cout << "Posicion eliminada exitosamente\n";
                break;
            }
            case 3: {
                try {
                    info = list.get(list.getFirst());
                    std::cout << "El numero en la primera posicion es " << info << "\n";
                } catch (std::exception &e) {
                    std::cout << "La lista esta vacia\n";
                }
                break;
            }
            case 4: {
                try {
                    info = list.get(list.getLast());
                    std::cout << "El numero en la ultima posicion es " << info << "\n";
                } catch (std::exception &e) {
                    std::cout << "La lista esta vacia\n";
                }
                break;
            }
            case 5: {
                std::cout << "Ingrese el numero a buscar: ";
                std::cin >> info;

                aux = list.find(info);
                if (aux != nullptr)
                    std::cout << "Se ha encontrado en la posicion " << list.getIndex(aux) << "\n";
                else
                    std::cout << "El numero no se encuentra en la lista\n";
                break;
            }
            case 6: {
                std::cout << "Ingrese la posicion a buscar: ";
```

```
                std::cin >> pos;

                aux = list.findByIndex(pos);
                try {
                    info = list.get(aux);
                    std::cout << "El numero en la posicion " << pos << " es " << info << "\n";
                } catch (std::exception &e) {
                    if (list.isEmpty())
                        std::cout << "La lista esta vacia\n";
                    else
                        std::cout << "La posicion no existe\n";
                }
                break;
            }
            case 7: {
                std::cout << list;
                std::cout << "\n";
                break;
            }
            case 0: {
                break;
            }
            default: {
                std::cout << "Opcion invalida";
                break;
            }

        }
    } while (opcion != 0);
}
```

Main.cpp//Menú:

En las primeras 3 imágenes se muestra el menú del programa el cual nos da 8 posibles opciones para ejecutar las cuales son: insertar, eliminar, mostrar el primer elemento, mostrar el último elemento, buscar, recuperar, imprimir y salir del programa.

```cpp
#include "node.h"
#include "list.h"

node::node() {
    next = nullptr;
    prev = nullptr;
}

std::ostream &operator<<(std::ostream &out, node &node) {
    out << node.info;
    return out;
}
```

Node.cpp//

En esta imagen se muestra como se define el elemento declarada en el apartado "public" de la clase node y se integra la sobrecarga del operador "<<".

```
#ifndef LISTA_CIRCULAR_DOBLEMENTE_LIGADA_CON_ENCABEZADO_NODE_H
#define LISTA_CIRCULAR_DOBLEMENTE_LIGADA_CON_ENCABEZADO_NODE_H

#include "ostream"

class node {
private:
    friend class list;

    int info;
    node *next;
    node *prev;
public:
    node();

    friend std::ostream &operator<<(std::ostream &, node &);
};


#endif //LISTA_CIRCULAR_DOBLEMENTE_LIGADA_CON_ENCABEZADO_NODE_H
```

Node.h//

En esta imagen se muestra la clase llamada "node" la cual tienen una clase friend llamada "list" y otra donde se sobrecarga el operador "<<".

1

```cpp
#include "list.h"

list::list() {
    first = nullptr;
    header = nullptr;
    last = nullptr;
    size = 0;
}

bool list::isEmpty() {
    return first == nullptr;
}

void list::add(int element, node *position) {
    node *aux = new node;
    aux->info = element;
    if (isEmpty()) {
        aux->next = aux;
        aux->prev = aux;
        first = aux;
    } else {
        if (position == nullptr) {
            aux->next = first;
            aux->prev = last;
            last->next = aux;
            first = aux;
        } else {
            aux->prev = position;
            aux->next = position->next;
            position->next->prev = aux;
            position->next = aux;
        }
    }
    if (aux->next == first)
        last = aux;
    size++;
}

void list::remove(node *position) {
    if (isEmpty() || position == nullptr) {
        return;
    }
    if (position->next == position) {
        first = nullptr;
        header = nullptr;
        last = nullptr;
    } else {
        position->prev->next = position->next;
        position->next->prev = position->prev;
        if (position == first)
            first = first->next;
        if (position == last)
            last = position->prev;
    }
    size--;
    delete position;
}

node *list::getFirst() {
    return first;
}

node *list::getLast() {
    return last;
}

node *list::previous(node *postition) {
    if (isEmpty() || postition == nullptr) {
        return nullptr;
    }
    return postition->prev;
}

node *list::next(node *postition) {

    if (isEmpty() || postition == nullptr) {
        return nullptr;
    }
    return postition->next;
}

int list::get(node *position) {
    if (isEmpty() || position == nullptr) {
        throw "Datos Insuficientes";
    } else {
        return position->info;
    }
}

node *list::find(int element) {
    if (isEmpty())
        return nullptr;
    node *aux = first;
    do {
        if (aux->info == element)
            return aux;
        aux = aux->next;
    } while (aux != first);
    return nullptr;
}
```

2

3

```cpp
list::~list() {
    if (isEmpty()) {
        return;
    }
    node *end = first;
    do {
        header = first;
        first = first->next;
        delete header;
    } while (first != end);
    first = nullptr;
    header = nullptr;
    last = nullptr;
    size = 0;
}

std::ostream &operator<<(std::ostream &out, list &list) {
    if (list.isEmpty()) {
        out << "La lista esta vacia";
    } else {
        node *aux = list.getFirst();
        do {
            out << list.get(aux) << ((list.next(aux) != list.getFirst()) ? ", " : "");
            aux = list.next(aux);
        } while (aux != list.getFirst());
    }
    return out;
}

node *list::findByIndex(int index) {
    node *aux = first;
    do{
        if (index == 0)
            return aux;
        aux = aux->next;
        index--;
    }while (aux != first);
    return nullptr;
}

int list::getIndex(node *index) {
    int i = 0;
    node *aux = first;
    do {
        if (aux == index)
            return i;
        aux = aux->next;
        i++;
    }while (aux != first);
    return -1;
}
```

List.cpp//

En este apartado se define el funcionamiento de cada una de las opciones que le permiten ejecutar al usuario desde el menú principal, tales opciones declaradas en la clase list.

```
#ifndef LISTA_CIRCULAR_DOBLEMENTE_LIGADA_CON_ENCABEZADO_LIST_H
#define LISTA_CIRCULAR_DOBLEMENTE_LIGADA_CON_ENCABEZADO_LIST_H

#include "node.h"

class list {
private:
    node *first;
    node *header;
    node *last;
    int size;
public:
    list();
    ~list();
    bool isEmpty();
    void add(int element, node *position);
    void remove(node *position);
    node *getFirst();
    node *getLast();
    node *previous(node *postition);
    node *next(node *postition);
    int get(node *position);
    node *find(int element);
    friend std::ostream &operator<<(std::ostream &, list &);
    node *findByIndex(int index);
    int getIndex(node *index);
};


#endif //LISTA_CIRCULAR_DOBLEMENTE_LIGADA_CON_ENCABEZADO_LIST_H
```

List.h//

En esta imagen se muestra la clase "list" la cual cuenta con las 8 opciones ejecutables por el usuario desde el menú principal.

Capturas de pantalla demostrando el funcionamiento de programas:

Compile Result

```
Numero agregado exitosamente
1.- Insertar
2.- Eliminar
3.- Primer Elemento
4.- Ultimo Elemento
5.- Busqueda
6.- Recupera
7.- Imprime
0.- Salir
1
Ingrese el numero a insertar: 100
Ingrese la posicion en la que desea
 insertar: 2
Numero agregado exitosamente
```

Compile Result

```
1.- Insertar
2.- Eliminar
3.- Primer Elemento
4.- Ultimo Elemento
5.- Busqueda
6.- Recupera
7.- Imprime
0.- Salir
1
Ingrese el numero a insertar: 70
Ingrese la posicion en la que desea
 insertar: 1
Numero agregado exitosamente
```

Compile Result

```
1.- Insertar
2.- Eliminar
3.- Primer Elemento
4.- Ultimo Elemento
5.- Busqueda
6.- Recupera
7.- Imprime
0.- Salir
1
Ingrese el numero a insertar: 25
Ingrese la posicion en la que desea
 insertar: 0
Numero agregado exitosamente
```

```
1.- Insertar
2.- Eliminar
3.- Primer Elemento
4.- Ultimo Elemento
5.- Busqueda
6.- Recupera
7.- Imprime
0.- Salir
2
Ingrese la posicion que desea elimi
nar: 1
Posicion eliminada exitosamente
```

Compile Result

```
1.- Insertar
2.- Eliminar
3.- Primer Elemento
4.- Ultimo Elemento
5.- Busqueda
6.- Recupera
7.- Imprime
0.- Salir
3
El numero en la primera posicion es
 100
```

Compile Result

```
1.- Insertar
2.- Eliminar
3.- Primer Elemento
4.- Ultimo Elemento
5.- Busqueda
6.- Recupera
7.- Imprime
0.- Salir
4
El numero en la ultima posicion es
25
```

```
1.- Insertar
2.- Eliminar
3.- Primer Elemento
4.- Ultimo Elemento
5.- Busqueda
6.- Recupera
7.- Imprime
0.- Salir
5
Ingrese el numero a buscar: 25
Se ha encontrado en la posicion 2
```

Compile Result

```
1.- Insertar
2.- Eliminar
3.- Primer Elemento
4.- Ultimo Elemento
5.- Busqueda
6.- Recupera
7.- Imprime
0.- Salir
6
Ingrese la posicion a buscar: 2
El numero en la posicion 2 es 25
```

Compile Result

```
Ingrese el numero a buscar: 25
Se ha encontrado en la posicion 2
1.- Insertar
2.- Eliminar
3.- Primer Elemento
4.- Ultimo Elemento
5.- Busqueda
6.- Recupera
7.- Imprime
0.- Salir
7
100, 70, 25
```

```
1.- Insertar
2.- Eliminar
3.- Primer Elemento
4.- Ultimo Elemento
5.- Busqueda
6.- Recupera
7.- Imprime
0.- Salir

0
proot info: vpid 1: terminated with
 signal 6

[Process completed (code 255) - pre
ss Enter]
```