**Actividad 14 – Pila y Cola**

**David Madrid Nápoles**

**Estructura de datos I**

# Lineamientos de evaluación

* El programa corre sin errores.
* Se implemento la clase Cola y Pila con sus métodos: push, pop, front, back, size y empty.
* En la implementación de Cola se hace uso de ListaDoblementeEnlazada como una fifo.
* En la implementación de Pila se hace uso de ListaDoblementeenlazada como una lifo.
* Se llevaron a cabo los procedimientos solicitados para realizar las capturas de pantalla como evidencia.

# Desarrollo

## Programa principal (salida main.exe)

Text

Description automatically generated

# Conclusiones

Me pareció fácil la implementación de Cola y Pila ya que los métodos requeridos ya estaban implementados en la lista doblemente ligada y solo se “heredaron”. Me pareció interesante el uso de FIFO y LIFO, estas maneras de utilizar las listas dinámicas me resultaron familiares y fáciles de asimilar.

# Referencias

<https://www.youtube.com/watch?v=235yQq2P-wc&feature=emb_imp_woyt>, Pila, Michel Davalos Boites.

<https://www.youtube.com/watch?v=235yQq2P-wc&feature=emb_imp_woyt>, Pila, Michel Davalos Boites.

# Código

//main.cpp

#include <iostream>

#include "Pila.h"

#include "Cola.h"

using namespace std;

int main() {

    Cola<int> cola;

    cola.push(-1); *// encolar*

    cola.push(0); *// encolar*

    cola.push(1); *// encolar*

    cola.push(2); *// encolar*

    cola.push(3); *// encolar*

    cout << \*cola.front() << endl; *// frente de la cola*

    cout << \*cola.back() << endl; *// final de la cola*

    cola.pop(); *// desencolar*

    cout << cola.size() << endl; *// imprimir la candidad de elementos en la cola*

    Pila<int> pila;

    while (!cola.empty()) *// ciclo para desencolar los elementos*

    {

        int \*e = cola.front(); *// respaldo del frente de la cola*

        if (e != nullptr) { *// si no es un puntero a nulo*

            pila.push(\*e); *// apilamos el entero*

        }

        cola.pop(); *// desencolar*

    }

    pila.push(10); *// apilar*

    cout << pila.size() << endl; *// imprimir la candidad de elementos en la pila*

    while (!pila.empty()) *// ciclo para desapilar los elementos*

    {

        int \*e = pila.top(); *// respaldo del tope de la pila*

        if (e != nullptr) { *// si no es un puntero nulo*

            cout << \*e << endl; *// imprime el tope de la pila*

        }

        pila.pop(); *// desapilar*

    }

    cout << pila.size() << endl; *// imprimir la candidad de elementos en la pila*

    return 0;

}

//Pila.h

#ifndef PILA\_H

#define PILA\_H

#include "ListaDoblementeLigada.h"

template <class T>

class Pila

{

private:

    ListaDoblementeLigada<T> lista;

    public:

    Pila();

    ~Pila();

    void push(const T &*dato*);

    void pop();

    T\* top();

    size\_t size();

    bool empty();

};

template <class T>

Pila<T>::Pila()

{

}

template <class T>

Pila<T>::~Pila()

{

}

template <class T>

void Pila<T>::push(const T &*dato*) {

    lista.push\_back(dato);

}

template <class T>

void Pila<T>::pop() {

    lista.pop\_back();

}

template <class T>

T\* Pila<T>::top() {

    return lista.back();

}

template <class T>

size\_t Pila<T>::size() {

    return lista.size();

}

template <class T>

bool Pila<T>::empty() {

    return lista.empty();

}

#endif

//Cola.h

#ifndef COLA\_H

#define COLA\_H

#include "ListaDoblementeLigada.h"

template <class T>

class Cola

{

private:

    ListaDoblementeLigada<T> lista;

public:

    Cola();

    ~Cola();

    void push(const T &*dato*);

    void pop();

    T\* front();

    T\* back();

    size\_t size();

    bool empty();

};

template <class T>

Cola<T>::Cola()

{

}

template <class T>

Cola<T>::~Cola()

{

}

template <class T>

void Cola<T>::push(const T &*dato*) {

    lista.push\_back(*dato*);

}

template <class T>

void Cola<T>::pop() {

    lista.pop\_front();

}

template <class T>

T\* Cola<T>::front() {

    return lista.front();

}

template <class T>

T\* Cola<T>::back() {

    return lista.back();

}

template <class T>

size\_t Cola<T>::size() {

    return lista.size();

}

template <class T>

bool Cola<T>::empty() {

    return lista.empty();

}

#endif

//ListaDoblementeligada.h

#ifndef LISTADOBLEMENTELIGADA

#define LISTADOBLEMENTELIGADA

#include <iostream>

using namespace std;

template <class T>

class ListaDoblementeLigada

{

private:

    struct Nodo

    {

        T dato;

        Nodo \*sig;

        Nodo \*ant;

        Nodo(const T &*dato*, Nodo \**sig* = nullptr, Nodo \**ant* = nullptr)

            : dato(*dato*), sig(*sig*), ant(*ant*) {}

    };

    Nodo \*head;

    Nodo \*tail;

    size\_t cont;

public:

    ListaDoblementeLigada();

    ~ListaDoblementeLigada();

    bool empty();

    void push\_front(const T &*dato*);

    void push\_back(const T &*dato*);

    void pop\_front();

    void pop\_back();

    void insert(const T &*dato*, size\_t *p*);

    void erase(size\_t *p*);

    T \*find(const T &*dato*);

    void remove\_if(const T &*dato*);

    void print();

    void print\_reverse();

    T \*front();

    T \*back();

    size\_t size();

    ListaDoblementeLigada &operator<<(const T &*dato*)

    {

        push\_back(*dato*);

        return \*this;

    }

    T \*operator[](size\_t *p*)

    {

        size\_t pos = 0;

        Nodo \*temp = head;

        while (temp != nullptr)

        {

            if (*p* = pos)

            {

                return &temp->dato;

            }

            temp = temp->sig;

            pos++;

            return nullptr;

        }

    }

};

template <class T>

ListaDoblementeLigada<T>::ListaDoblementeLigada()

{

    head = nullptr;

    tail = nullptr;

    cont = 0;

}

template <class T>

ListaDoblementeLigada<T>::~ListaDoblementeLigada()

{

    while (!empty())

    {

        pop\_front();

    }

}

template <class T>

bool ListaDoblementeLigada<T>::empty()

{

    return cont == 0;

}

template <class T>

void ListaDoblementeLigada<T>::push\_front(const T &*dato*)

{

    Nodo \*nodo = new Nodo(*dato*, head);

    if (cont == 0)

    {

        head = nodo;

        tail = nodo;

    }

    else

    {

        head->ant = nodo;

        head = nodo;

    }

    cont++;

}

template <class T>

void ListaDoblementeLigada<T>::push\_back(const T &*dato*)

{

    Nodo \*nodo = new Nodo(*dato*, nullptr, tail);

    if (cont == 0)

    {

        head = nodo;

        tail = nodo;

    }

    else

    {

        tail->sig = nodo;

        tail = nodo;

    }

    cont++;

}

template <class T>

void ListaDoblementeLigada<T>::pop\_front()

{

    if (empty())

    {

        cout << "Lista vacia..." << endl;

    }

    else if (cont == 1)

    {

        delete head;

        head == nullptr;

        tail == nullptr;

        cont--;

    }

    else

    {

        Nodo \*temp = head->sig;

        head->sig->ant = nullptr;

        delete head;

        head = temp;

        cont--;

    }

}

template <class T>

void ListaDoblementeLigada<T>::pop\_back()

{

    if (empty())

    {

        cout << "Lista vacia..." << endl;

    }

    else if (cont == 1)

    {

        delete tail;

        tail == nullptr;

        head == nullptr;

        cont--;

    }

    else

    {

        Nodo \*temp = tail->ant;

        temp->sig = nullptr;

        delete tail;

        tail = temp;

        cont--;

    }

}

template <class T>

size\_t ListaDoblementeLigada<T>::size()

{

    return cont;

}

template <class T>

void ListaDoblementeLigada<T>::print()

{

    Nodo \*temp = head;

    while (temp != nullptr)

    {

        cout << temp->dato << endl;

        temp = temp->sig;

    }

}

template <class T>

void ListaDoblementeLigada<T>::print\_reverse()

{

    Nodo \*temp = tail;

    while (temp != nullptr)

    {

        cout << temp->dato << endl;

        temp = temp->ant;

    }

}

template <class T>

T \*ListaDoblementeLigada<T>::front()

{

    if (empty())

    {

        return nullptr;

    }

    else

    {

        return &head->dato;

    }

}

template <class T>

T \*ListaDoblementeLigada<T>::back()

{

    if (empty())

    {

        return nullptr;

    }

    else

    {

        return &tail->dato;

    }

}

template <class T>

void ListaDoblementeLigada<T>::insert(const T &*dato*, size\_t *p*)

{

    if (*p* >= cont)

    {

        cout << *p* << " es una posicion no valida" << endl;

    }

    else if (*p* == 0)

    {

        push\_front(*dato*);

    }

    else

    {

        Nodo \*temp = head->sig;

        size\_t pos = 1;

        while (temp != nullptr)

        {

            if (*p* == pos)

            {

                Nodo \*nodo = new Nodo(*dato*);

                nodo->sig = temp;

                nodo->ant = temp->ant;

                temp->ant->sig = nodo;

                nodo->sig->ant = nodo;

                cont++;

                break;

            }

            temp = temp->sig;

            pos++;

        }

    }

}

template <class T>

void ListaDoblementeLigada<T>::erase(size\_t *p*)

{

    if (*p* >= cont)

    {

        cout << *p* << " es una posicion no valida" << endl;

    }

    else if (*p* == 0)

    {

        pop\_front();

    }

    else if (*p* == (cont - 1))

    {

        pop\_back();

    }

    else

    {

        Nodo \*temp = head->sig;

        size\_t pos = 1;

        while (temp != nullptr)

        {

            if (*p* == pos)

            {

                temp->ant->sig = temp->sig;

                temp->sig->ant = temp->ant;

                delete temp;

                cont--;

                break;

            }

            temp = temp->sig;

            pos++;

        }

    }

}

template <class T>

T \*ListaDoblementeLigada<T>::find(const T &*dato*)

{

    Nodo \*temp = head;

    bool encontrado = false;

    while (temp != nullptr)

    {

        if (temp->dato == *dato*)

        {

            encontrado == true;

            return &temp->dato;

        }

        temp = temp->sig;

    }

    if (!encontrado)

    {

        return nullptr;

    }

}

template <class T>

void ListaDoblementeLigada<T>::remove\_if(const T &*dato*)

{

    if(empty()){

        cout << "Lista vacia..." << endl;

    }else{

        Nodo \*temp = head;

        size\_t p = 0;

        while(temp->sig != nullptr){

            if(temp->dato == *dato*){

                if(p == 0){

                    pop\_front();

                    temp = head; *//Reinicia el cliclo y p ya es 0*

                }else{

                    erase(p);

                    temp = head; *//Reinicia el cliclo*

                    p = 0;

                }

            } else { *//Si no es igual, pasa al siguiente nodo*

                p++;

                temp = temp->sig;

            }

        }

*//Si llega aqui ya recorrio toda la lista*

        if(temp->dato == *dato*){

            pop\_back();

        }

    }

}

#endif