Actividad 12 – Lista doblemente ligada

David Madrid Nápoles

Estructura de datos I

Lineamientos de evaluación

- El programa corre sin errores.
- Se implemento la clase ListaDoblementeLigada con sus métodos:

  ListaDoblementeLigada();

  ~ListaDoblementeLigada();

  bool empty();

  void push_front(const T &dato);

  void push_back(const T &dato);

  void pop_front();

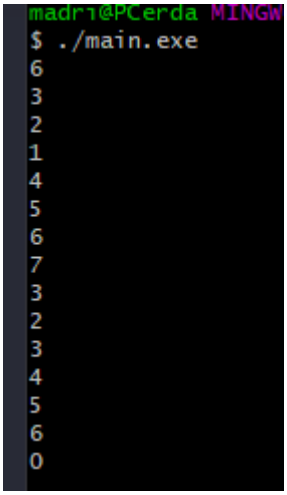
```
void pop_back();
void insert(const T &dato, size_t p);
void erase(size_t p);
T* find(const T &dato);
void print();
void print_reverse();
T *front();
T *back();
size_t size();
```

Se llevaron a cabo los procedimientos solicitados para realizar las capturas de pantalla como evidencia.

Desarrollo

Programa principal (salida main.exe)

Corriendo el main solicitado en la actividad:



Conclusiones

Sentí la lista doblemente ligada fácil de usar, sobre todo al momento de borrar un nodo en un punto medio el uso del valor ant fue muy útil para moverse entre los nodos.

Mi mayor dificultad fue haber confundido la actividad 13 con la 12, implementé ya varios de los métodos de ese video, solo faltándome remove_if, no la pude implementar, pero de la actividad 12 fue ya bastante sencillo el método find, para que retornara el nullptr me apoyé de una bandera que se hace verdadera cuando encuentra un valor, y dentro de un condicional lo retorna.

Referencias

https://www.youtube.com/watch?v=5VQj3Ep2RMI&t=260s, Lista Doblemente Ligada (I), Michel Davalos Boites.

https://www.youtube.com/watch?v=DOjuBkDDAMs, Lista Doblemente Ligada (II), Michel Davalos Boites.

Código

```cpp
//main.cpp

#include <iostream>
#include "ListaDoblementeLigada.h"

using namespace std;

int main() {
    ListaDoblementeLigada<int> enteros;

    enteros.push_front(1);           // insertar al inicio (frente)
    enteros.push_front(2);           // insertar al inicio (frente)
    enteros.push_front(3);           // insertar al inicio (frente)

    enteros << 4 << 5 << 6;          // insertar al final (cola)

    cout << enteros.size() << endl;  // imprime la cantidad de nodos (elementos)

    enteros.print();                 // imprime el elemento de cada nodo

    int *ptr_01 = enteros.find(1); // buscar un elemento
    if (ptr_01) {                    // si no es nulo
        *ptr_01 = 3;                 // cambiar valor
    }

    int *ptr_02 = enteros.find(0);   // buscar un elemento
    if (ptr_02 == nullptr) {         // si es nulo
        enteros << 0;                // inserta al final
    }

    cout << enteros.size() << endl;  // imprime la cantidad de nodos (elementos)

    enteros.print();                 // imprime el elemento de cada nodo

    return 0;
```

```
}
//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 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>
```

```cpp
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;
    }
}

#endif
```