Jorge Del Barco Garza
A01284234

Actividad 2.1: LinkedLists

Create

```
void LinkedList<T>::create(T data, int index)
{

    // mensaje de error si no se puede insertar en esa posición "index" el valor "data"
    if (index > size)
    {
        std::cout << "No hay suficientes elementos para insertar el valor " << data << " en la posicion: " << index << std::endl;
        return;
    }
    else if (index < 0)
    {
        std::cout << "Esa posicion no existe" << std::endl;
        return;
    }

    // Si se quiere insertar en la primera posición utilizamos la función que ya tenemos.
    if (index == 0)
    {
        addFirst(data);
        return;
    }
```

```
    // Variable que insertará en la linked list el valor "data". Una vez se hayan traversado "index" elementos.
    int pivot = 0;

    Node<T> *current = head;

    // Atravesamos la linked list hasta llegar a la posición deseada-1 para ocupar el índice exácto de la lista
    while (pivot < index - 1)
    {
        current = current->getNext();
        pivot = pivot + 1;
    }

    // El nuevo nodo apunta a donde apuntaba el nodo anterior.
    Node<T> *newNode = new Node<T>(data, current->getNext());

    // El nodo anterior apunta al nuevo nodo creado
    current->setNext(newNode);

    // incrementamos el tamaño de la lista
    size = size + 1;
}
```

Jorge Del Barco Garza
A01284234

Actividad 2.1

function create (T data, int index)

if index > size C_1 1

return

else if (index < 0) C_2 1

return

if index == 0 C_3 1

addFirst(data)
return

int pivot = 0 C_4 1

~~Node* current = head~~

while pivot < index - 1 $C_5 \sum^{index-1} t_j$

current = current -> getNext() $C_6 \sum^{index} t_j - 1$

pivot = pivot + 1 $C_7 \sum^{index} (t_j - 1)$

Node *newNode = new Node<T>(data, current -> getNext() C_8 1

current -> setNext(newNode) C_9 1

$$T(n) = C_1 + C_2 + C_3 + C_4 + C_5 \sum^{index} t_j + C_6 (\sum^{index} t_j - 1) + C_7 (\sum^{index} t_j - 1)$$

$$T(n) = C_1 + C_2 + C_3 + C_4 + C_5 \sum t_j + C_6 \sum t_j - C_6 + C_7 \sum t_j - C_7$$

$$T(n) = \sum^{index} t_j (C_5 + C_6 + C_7) = \boxed{O(n)}$$

Jorge Del Barco Garza
A01284234

Read

```cpp
template <class T>
T LinkedList<T>::read(int index)
{

    if (index > size - 1 || index < 0)
    {
        std::cout << "Esa posicion no existe" << std::endl;
        return -1;
    }

    // Variable que insertará en la linked list el valor "data". Una vez se hayan traversado "index" elementos.
    int pivot = 0;
    Node<T> *current = head;

    // Atravesamos la linked list hasta llegar a la posición deseada-1 para ocupar el índice exácto de la lista
    while (pivot < index)
    {
        current = current->getNext();
        pivot = pivot + 1;
    }

    return current->getData();
}
```

Jorge Del Barco Garza
A01284234

```
func read (int index)
if (index > size-1 or index < 0)        C1 | 1
    return -1;
int pivot = 0                           C2 | 1
Node * current = head                   C3 | 1
while pivot < index                     C4 | Σ_{pivot}^{index} tj
    current = current -> getNext()      C5 | Σ_{pivot}^{index} tj - 1
    pivot = pivot + 1                   C6 | Σ_{pivot}^{index} tj - 1
return current -> getData               C7 | 1
```

$$T(n) = C_1 + C_2 + C_3 + C_4 \Sigma t_j + C_5(\Sigma t_j - 1) + C_6(\Sigma t_j - 1) + C_7$$

$$T(n) = C_1 + C_2 + C_3 + C_4 \Sigma t_j + C_5 \Sigma t_j - C_5 + C_6 \Sigma t_j - C_6 + C_7$$

$$T(n) = (C_1 + C_2 + C_3 - C_5 - C_6 + C_7) + \Sigma t_j (C_4 + C_5 + C_6)$$

$$T(n) = a + \Sigma t_j b = \boxed{O(n)}$$

Jorge Del Barco Garza
A01284234

Update

```
void LinkedList<T>::update(T data, int index)
{

    if (index > size - 1 || index < 0)
    {
        std::cout << "Esa posicion no existe" << std::endl;
        return;
    }

    // Variable que insertará en la linked list el valor "data". Una vez se hayan traversado "index" elementos.
    int pivot = 0;
    Node<T> *current = head;

    // Atravesamos la linked list hasta llegar a la posición deseada-1 para ocupar el índice exácto de la lista
    while (pivot < index)
    {
        current = current->getNext();
        pivot = pivot + 1;
    }

    // Actualizamos el valor del elemento ubicado en la posición "index";
    current->setData(data);
}
```

Jorge Del Barco Garza
A01284234

```
func update (data, int index)
if (index > size -1 or index < 0)        C1
    return
int pivot = 0                            C2
Node * current = head                    C3
while (pivot < index)                    C4  Σ_{j=0}^{index} tj
    current = current -> getNext()       C5  Σ_{j=0}^{index} tj - 1
    pivot = pivot + 1                    C6  Σ_{j=0}^{index} tj - 1
current -> setData (data)                C7
```

$$T(n) = C_1 + C_2 + C_3 + C_4 \sum_{j=0}^{index} tj + C_5\left(\sum_{j=0}^{index} tj - 1\right) + C_6\left(\sum_{j=0}^{index} tj - 1\right) + C_7$$

$$T(n) = (C_1 + C_2 + C_3 + C_7 - C_5 - C_6) + \sum_{j=0}^{index} tj\,(C_4 + C_5 + C_6)$$

$$T(n) = a + \sum_{j=0}^{index} tj\,(b) = \boxed{O(n)}$$

Jorge Del Barco Garza
A01284234

Del

```cpp
void LinkedList<T>::del(int index)
{
    if (index > size - 1 || index < 0)
    {
        std::cout << "Esa posicion no existe" << std::endl;
        return;
    }

    // Si quiere borrar la cabeza
    if (index == 0)
    {
        delete head;

        // cambiamos la cabeza al siguiente nodo
        head = head->getNext();
        return;
    }
    // Variable que insertará en la linked list el valor "data". Una vez se hayan traversado "index" elementos.
    int pivot = 0;
    Node<T> *current = head;
    // Debemos tener referencia del nodo anterior para contectarlo con el ->getNext()->getNext()
    Node<T> *previousNode;

    // Atravesamos la linked list hasta llegar a la posición deseada para ocupar el elemento anterior al dado.
    while (pivot < index)
    {
        previousNode = current;
        current = current->getNext();
        pivot = pivot + 1;
    }
    previousNode->setNext(current->getNext());
    // Liberamos memoria del nodo eliminado
    delete current;
}
```

Jorge Del Barco Garza
A01284234

```
func del (int index)
if (index > size-1 or index < 0)   C1
    return
if index == 0
    delete head
    head = head->getNext()          C2
    return
int pivot = 0                       C3
Node* current=head                  C4
Node* previousNode                  C5
while(pivot < index)                C6 Σ tj^index
    previousNode = current          C7 Σ tj-1
    current = current->getNext()    C8 Σ tj-1
    pivot = pivot + 1               C9 Σ tj-1
previousNode -> setNext(current->getNext)   C10
delete current;                     C11
```

$$T(n) = C_1 + C_2 + C_3 + C_4 + C_5 + C_6 t_j + C_7 t_j - C_7 + C_8 t_j - C_8 + C_9 t_j - C_9 + \dots$$

$$T(n) = (C_1 + C_2 + C_3 + C_4 + C_5 - C_7 - C_8 - C_9 + C_{10} + C_{11}) + t_j (C_6 + C_7 + C_8 + C_9)$$

$$a + t_j(b) = O(n)$$