Listas en C++

Algo2 – C1 – 2020

Motivación

- Nos gustaría evitar la complejidad de pedir O(n) posiciones de memoria al mismo tiempo como hicimos en Vector<T>

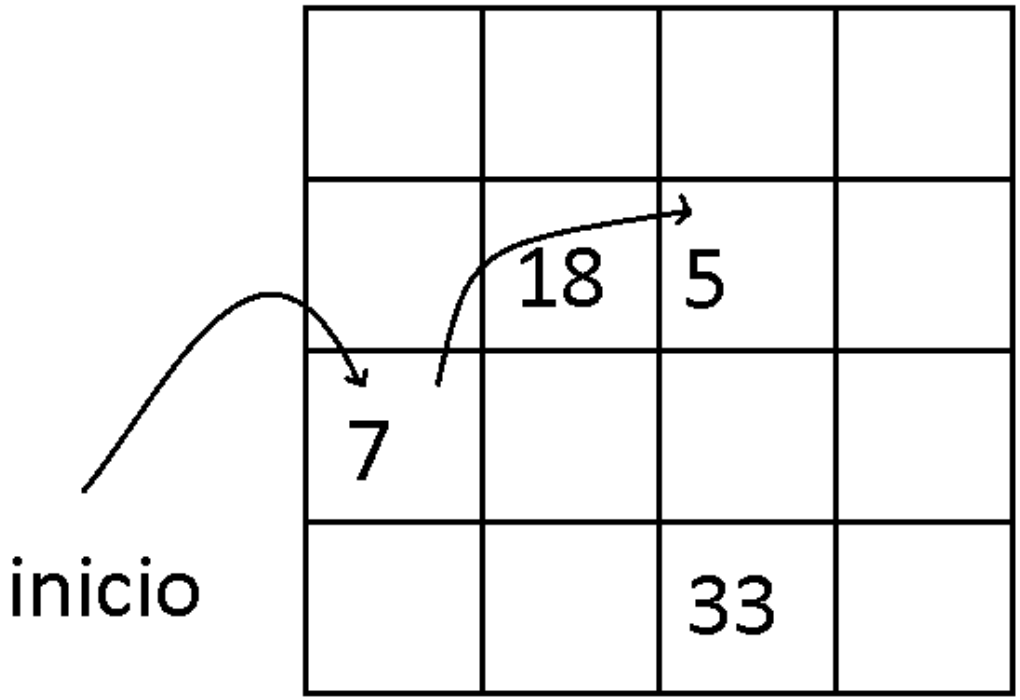
Motivación

- Nos gustaría implementar un TAD que solicite la memoria por demanda a medida que se agregan nuevos elementos en O(1)

¿Cómo podemos hacerlo?

Diseño

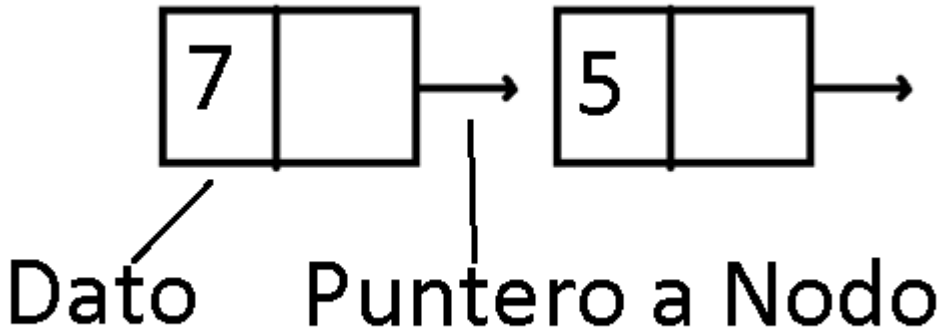
- ¡Pero no pueden estar contiguos y que sea O(1) a la vez!
- Por ejemplo si agregamos 7 y 5

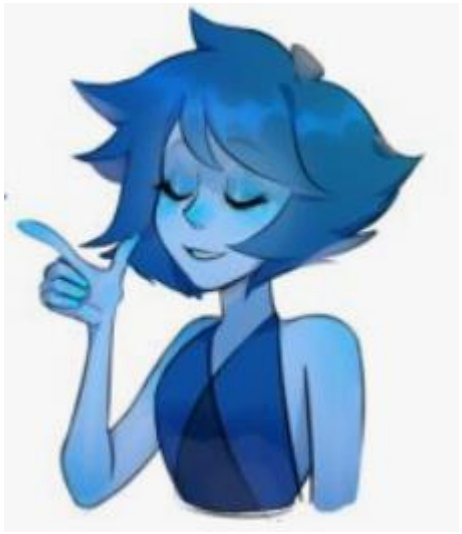


Nodo

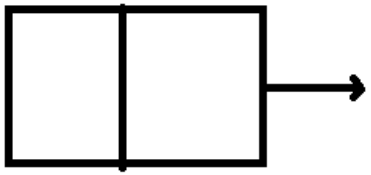
- La estructura clásica para modelar esto es el nodo

Nodo





Nodo



- Node será la primer estructura recursiva que vamos a ver

```
#include <iostream>
#include <cstdlib>

class Node
{
public:
   Node* next;
   int data;
};

using namespace std;
```

←Puntero a Nodo

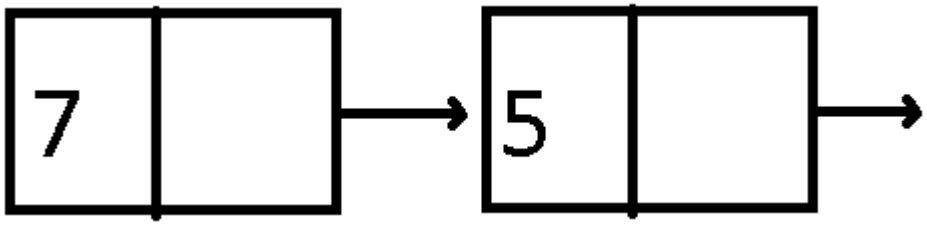
← Consideración técnica que nos permitirá simplificar el código

Lista

- ¿Podremos encapsular este comportamiento en un TAD que se comporte como std::list<T>?

```
std::list<int> l = { 7, 5 };
```

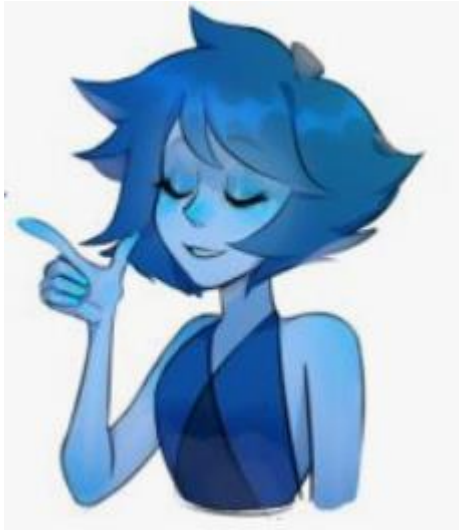
Lista



- Encabezado

```
class LinkedList
{
public:
    int length;
    Node* head;

    LinkedList();
    ~LinkedList();
    void add(int data);
    void print();
};
```



Lista

- Encabezado y constructor

```
class LinkedList
{
public:
   int length;
   Node* head;

   LinkedList();
   ~LinkedList();
   void add(int data);
   void print();
};
```

```
LinkedList::LinkedList(){
   length = 0;
}
```

Lista

- Encabezado y constructor

```
class LinkedList
{
public:
    int length;
    Node* head;

    LinkedList();
    ~LinkedList();
    void add(int data);
    void print();
};
```

```
LinkedList::LinkedList(){
    length = 0;
}
```

¡Dependemos de los constructores default de Node e int!

Notar que "length = 0" no es la 1er instrucción! ¿Cual es entonces?

Lista

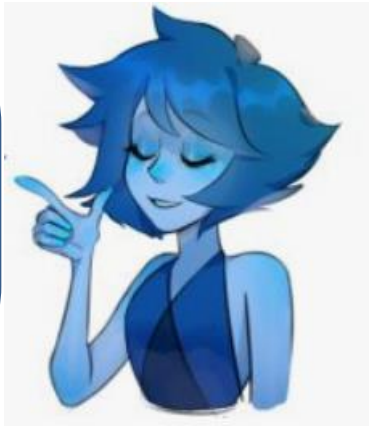
- Encabezado y constructor

```
class LinkedList
{
public:
    int length;
    Node* head;

    LinkedList();
    ~LinkedList();
    void add(int data);
    void print();
};
```

```
LinkedList::LinkedList(){
    : length(0), head(NULL) {}
}
```

Es mejor inicializar las variables para no abusar de los ***valores por defecto***



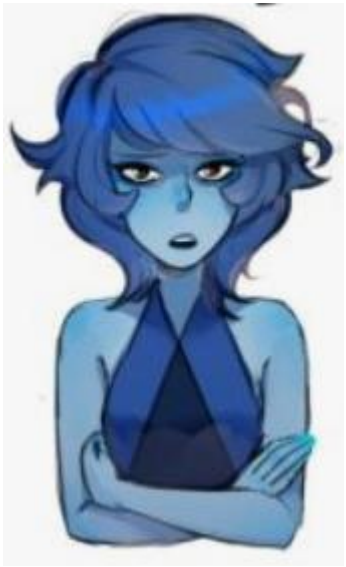
Un paso a la vez

- Implementaremos una lista de enteros, para resaltar las operaciones new/delete sin lidiar con las particularidades de Templates.

Agregar

Paso por paso.

```
void LinkedList::add(int data){
    …
}
```

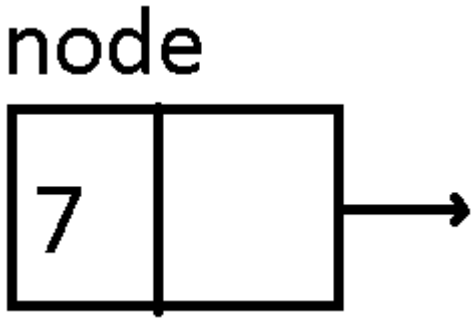


¿Qué es lo primero que tenemos que hacer?

Agregar

Paso por paso.

```
void LinkedList::add(int data){
    Node* node = new Node();
    node->data = data;
    …
}
```

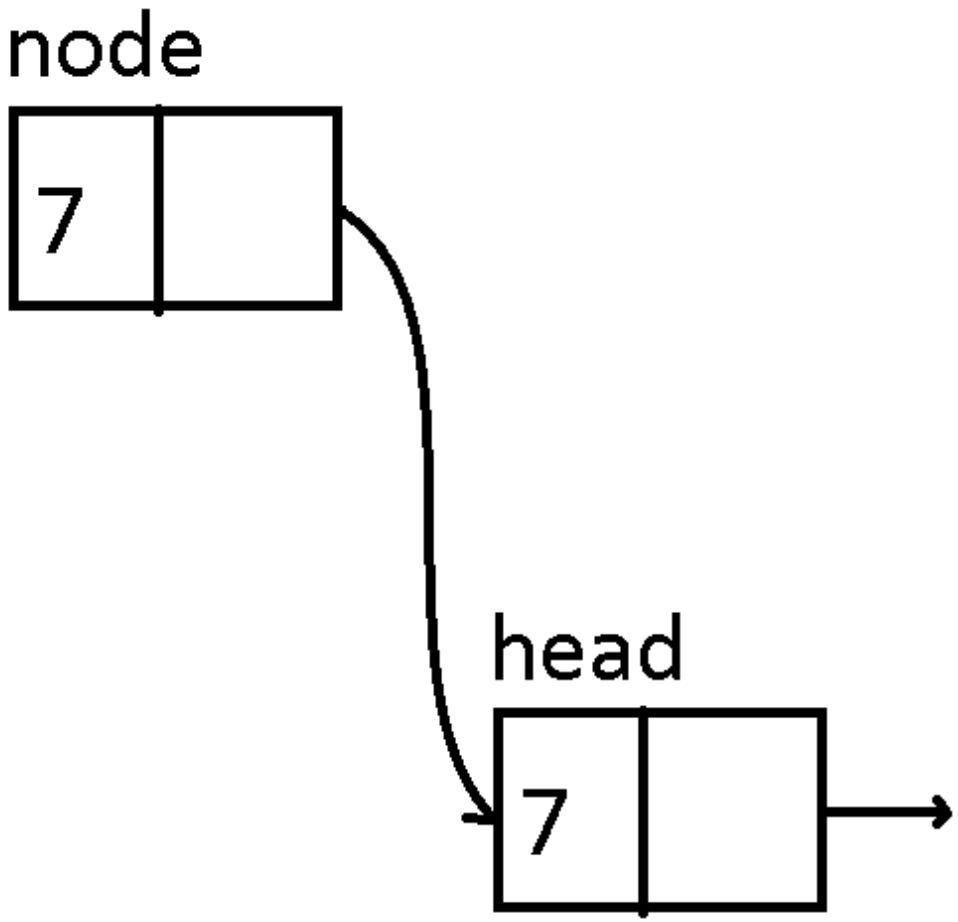


Reservamos memoria para el nuevo nodo

Agregar

Ahora hay que “enganchar” node al resto de la lista

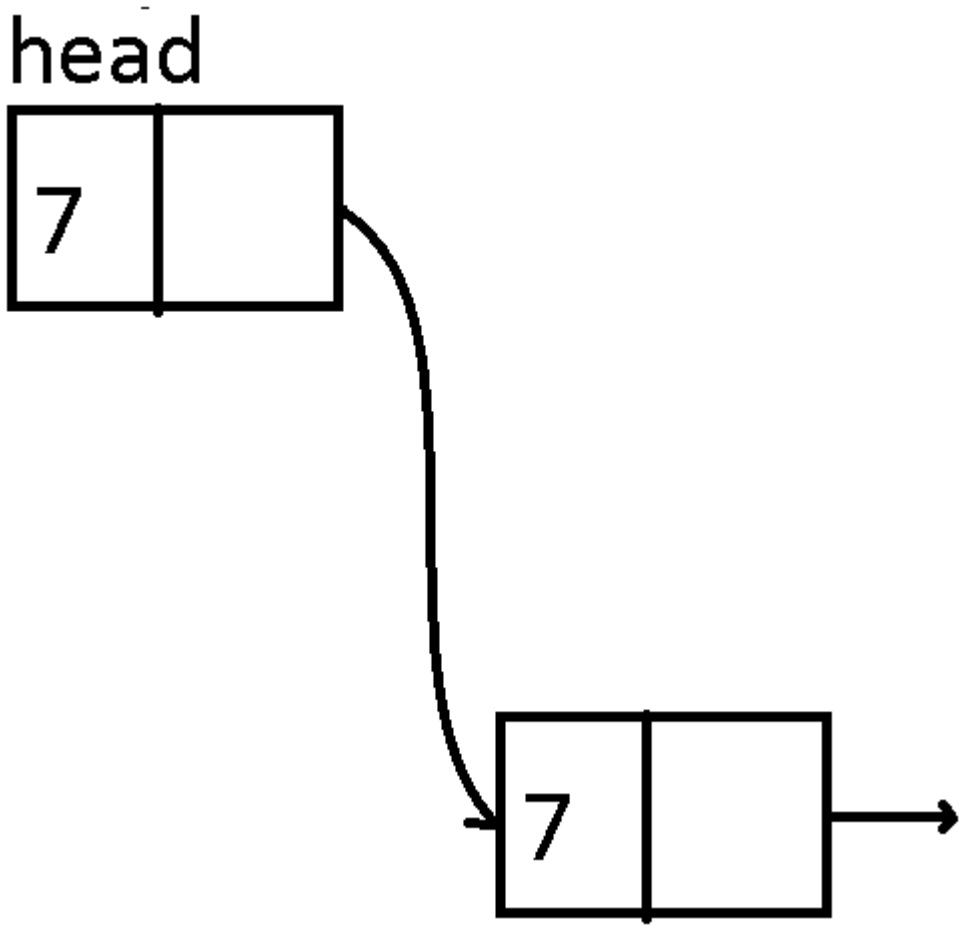
```
void LinkedList::add(int data){
    Node* node = new Node();
    node->data = data;
    node->next = this->head;
    this->head = node;
    this->length++;
}
```



Agregar

Ahora hay que “enganchar” node al resto de la lista

```
void LinkedList::add(int data){
    Node* node = new Node();
    node->data = data;
    node->next = this->head;
    this->head = node;
    this->length++;
}
```



Agregar

Ahora hay que preservar el IREP*

```
void LinkedList::add(int data){
    Node* node = new Node();
    node->data = data;
    node->next = this->head;
    this->head = node;
    this->length++;
}
```

*Parte del IREP que nos interesa:
length = #recorrerAMano(head)

Destructor

¡Es muy fácil!

```
LinkedList::~LinkedList()
{
        delete[] head
}
```

Destructor

- ¡Pero no funciona!
- Porque head no es un vector (ni tampoco una lista).
- Head es un puntero a nodo (al primer nodo).

Enfoque:

¡Tengo que hacer #length delete's, de lo contrario, pierdo memoria!

Destructor

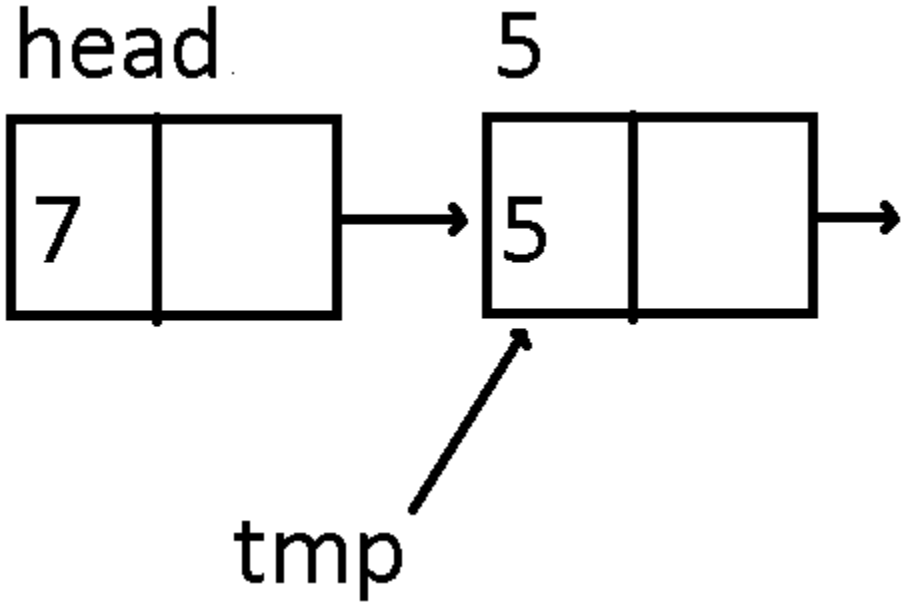
¡Ojo con los punteros!

```
LinkedList::~LinkedList()
{
        Node<T>* temp = head->next;
        while(temp != NULL)
        {
                temp = temp->next;
                delete(head);
                head = temp;
        }
}
```

Destructor

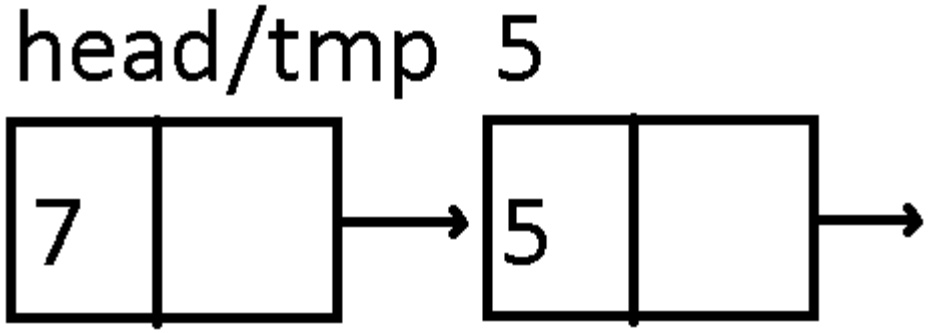
¡Ojo con los punteros!

```
LinkedList::~LinkedList()
{
        Node<T>* temp = head->next;
        while(temp != NULL)
        {
                temp = temp->next;
                delete(head);
                head = temp;
        }
}
```



Destructor

Paso por paso.

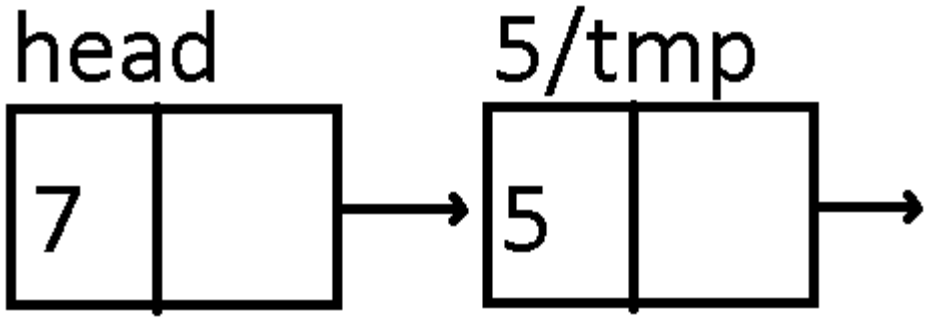



```
Node<T>* temp = head; (1)
while(temp != NULL)
{
        temp = temp->next;  (2)
        delete(head); (3)
        head = temp; (4)
}
```

Destructor

Paso por paso.

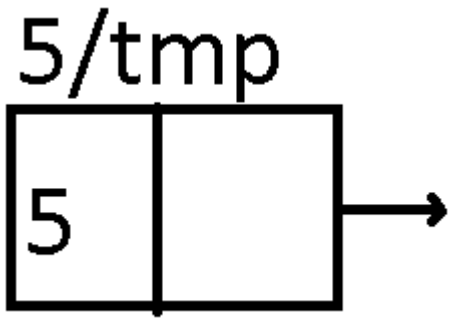
```
Node<T>* temp = head; (1)
while(temp != NULL)
{
        temp = temp->next;  (2)
        delete(head); (3)
        head = temp; (4)
}
```



Destructor

Paso por paso.

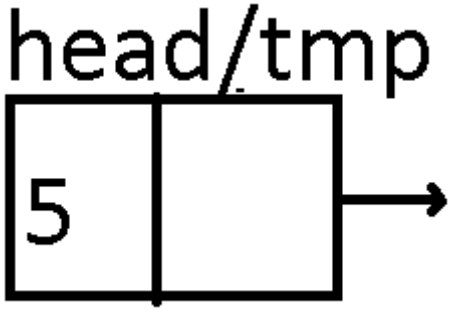
```
Node<T>* temp = head; (1)
while(temp != NULL)
{
        temp = temp->next;  (2)
        delete(head); (3)
        head = temp; (4)
}
```



Destructor

Paso por paso.

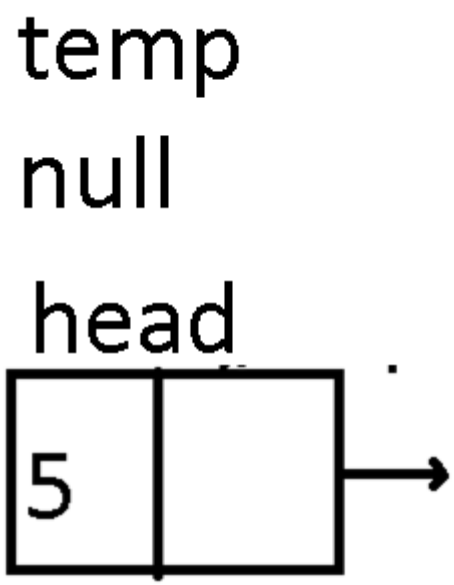
```
Node<T>* temp = head; (1)
while(temp != NULL)
{
        temp = temp->next;  (2)
        delete(head); (3)
        head = temp; (4)
}
```



Destructor

Paso por paso.

```
Node<T>* temp = head; (1)
while(temp != NULL)
{
        temp = temp->next;  (2 bis)
        delete(head); (3)
        head = temp; (4)
}
```



Lista: Otro destructor

- Destructor2

```
Lista::Lista() {
        _primero = NULL;
}
Lista::~Lista() {
        _destruir();
}
void Lista::_destruir() {
        while (_primero != NULL) {
                sacarPrimero();
        }
}
```

```
void Lista::sacarPrimero() {
        Nodo* p = _primero;
        _primero =
                _primero->siguiente;
        delete p;
}
```

Constructor de Nodo

- Encabezado

```
#include <iostream>
using namespace std;

class Lista {
public:
        Lista();
        ~Lista();
void agregar (const int& x);
```

¡Hacer un constructor que se adapte a nuestro caso de uso!

```
    private:
            void _destruir();

            struct Nodo {
            Nodo(const int& elem);
                    int valor;
                    Nodo* siguiente;
            };
            Nodo *_primero;
    };
```

Lista<T>: Nuevo agregar()

```
void Lista::agregar (const int& x) {
        Nodo* nuevo = new Nodo(x);
        nuevo->siguiente = _primero;
        _primero = nuevo;
}
```

Lista::Nodo::Nodo(const int& elem)
: valor(elem), prox(NULL) {};

Lista: Ejemplo

```
int main() {
        Lista s;
        s.agregar (40);
        s.agregar (50);
        s.agregar (60);
        cout << s << endl;
        return 0;
}
```