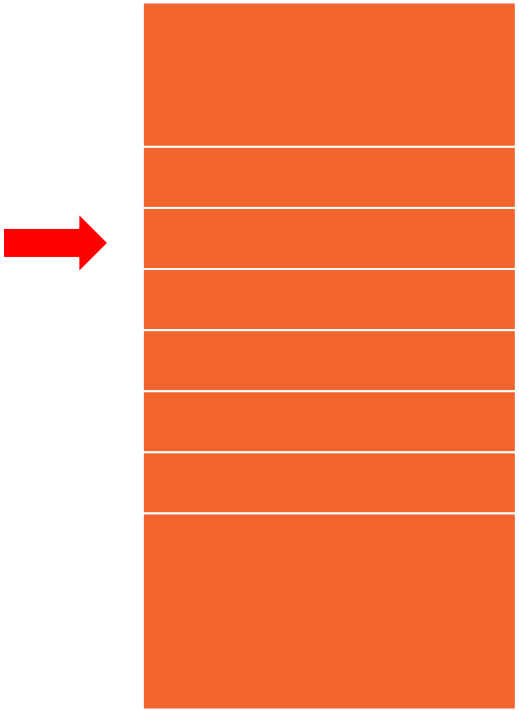
Estructuras de datos

Punteros

(pointer)

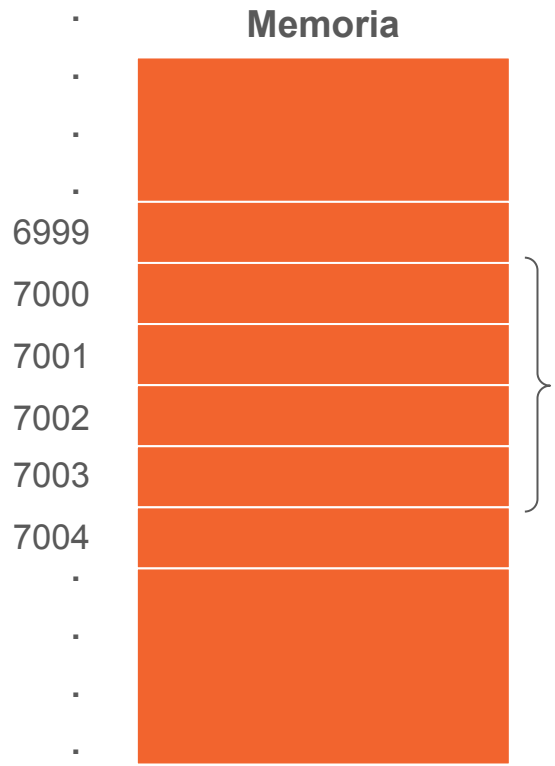
UTN San Nicolás
SEC. POSGRADO Y EDUCACIÓN CONTINUA

Punteros



Puntero: Variable que almacena una dirección de memoria

Punteros



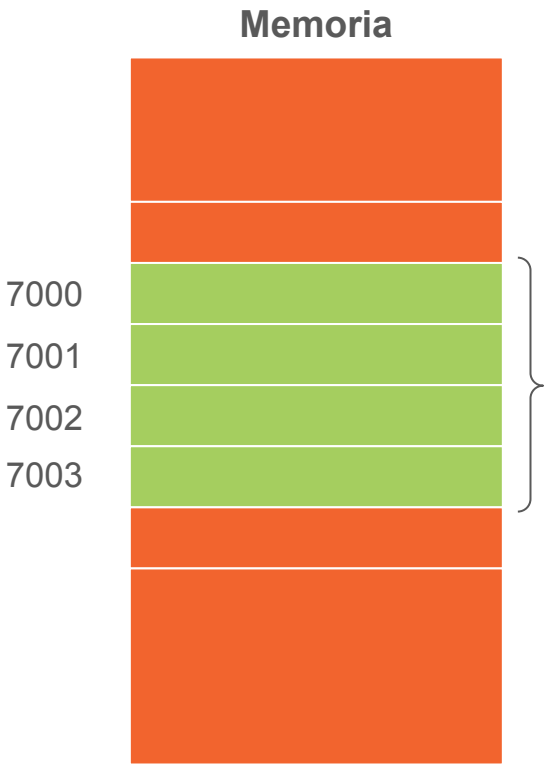
Puntero: Variable que almacena una dirección de memoria

Por ejemplo:

- Variable entera: int edad;

Declaración: int *p_edad;

Punteros



Puntero: Variable que almacena una dirección de memoria

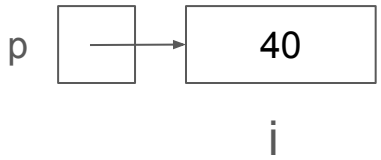
Por ejemplo:

- Variable entera: int edad;
- Ocupa 4 bytes en memoria
- Un puntero almacenará la dirección de memoria del primer byte (7000)

Declaración: int *p_edad;

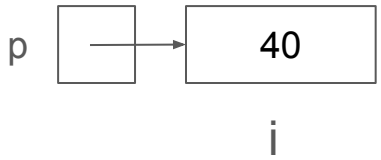
Los operadores de los punteros

- Referencia: &
  - Hace que un puntero apunte a una variable
  - Ejemplo:
    - int i = 40;
    - int *p = &i;



Los operadores de los punteros

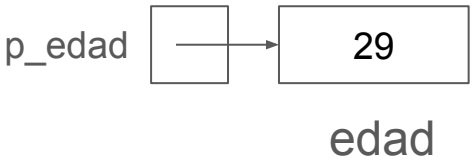
- Referencia: &
  - Hace que un puntero apunte a una variable
  - Ejemplo:
    - int i = 40;
    - int *p = &i;



- Indirección: *
  - Permite acceder al valor que hay en la memoria a la que apunta un puntero
  - Ejemplo: printf(“%d”, *p);

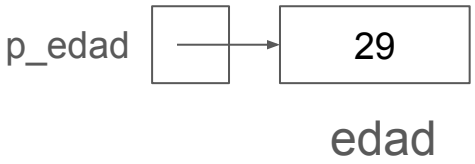
Trabajando con punteros

```
int edad, *p_edad;
edad = 29;
p_edad = &edad;
```

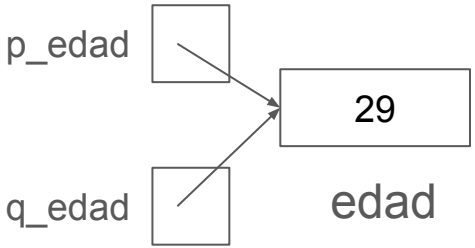


Trabajando con punteros

```
int edad, *p_edad;
edad = 29;
p_edad = &edad;
```

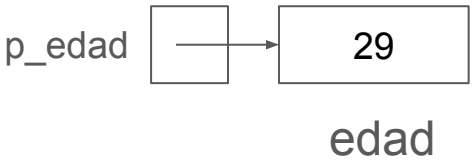



```
int *q_edad;
q_edad = p_edad;
```

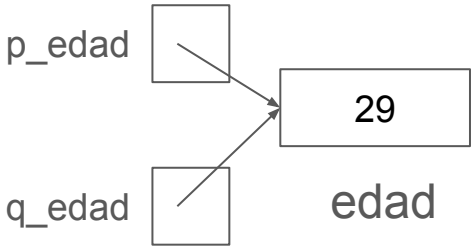


Trabajando con punteros

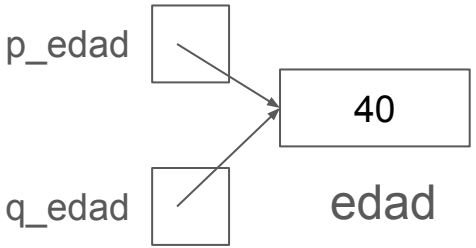
```
int edad, *p_edad;
edad = 29;
p_edad = &edad;
```




```
int *q_edad;
q_edad = p_edad;
```




```
*p_edad = 40;
```



Ejemplo de punteros

```
#include <iostream>

using namespace std;

int main()
{
    int edad, *p;
    printf("%i \n\n", edad);
    edad = 3;
    printf("%i \n\n", edad);
    p = &edad;
    int *q;
    q = p;
    printf("%i %i %p %i\n\n", edad, *p, p, *q, q);
    *p = 29;
    printf("%i %i %p %i\n\n", edad, *p, p, *q, q);
    return 0;
}
```

UTN San Nicolás
SEC. POSGRADO Y EDUCACIÓN CONTINUA

Punteros pasando parámetros por referencia

- Hacer nuestro programas más eficientes

Punteros pasando parámetros por referencia

- Hacer nuestro programas más eficientes
- Los punteros son útiles para simular un paso de parámetros por referencia

```
void max_min(int a[], int n, int *max, int *min)
{
    *max = *min = a [0];
    for (int i = 1; i < n; i++) {
        if (a[i] > *max)
            *max = a[i];
        else if (a[i] < *min)
            *min = a[i];
    }
}
```

Punteros pasando parámetros por referencia

- Hacer nuestro programas más eficientes
- Los punteros son útiles para simular un paso de parámetros por referencia
- Es mejor hacer copia de una dirección de memoria y su contenido a tener que copiar la variable por completo

Punteros pasando parámetros por referencia

- Hacer nuestro programas más eficientes
- Los punteros son útiles para simular un paso de parámetros por referencia
- Es mejor hacer copia de una dirección de memoria y su contenido a tener que copiar la variable por completo
- La palabra reservada CONST sirve para mantener el valor inmutable

```
void miFuncion(const int *p) {
    *p = 0; // no se puede
}
```

Structuras

(struct)

Definir una estructura

```
struct Automovil
{
    int m_iNumeroDeModelo;
    int m_iKilometraje;
    int m_iCilindrada;
};
```

Utilizar una estructura

```
Automovil      miAuto;
```

Utilizar una estructura

Automovil miAuto;

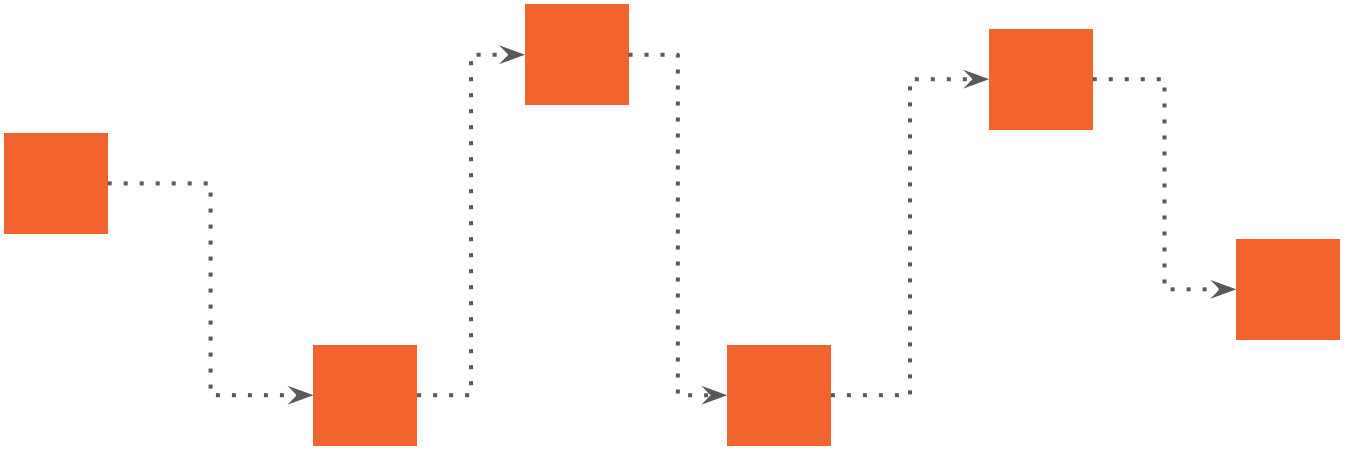
Tipo de dato Nombre de la variable

Utilizar una estructura

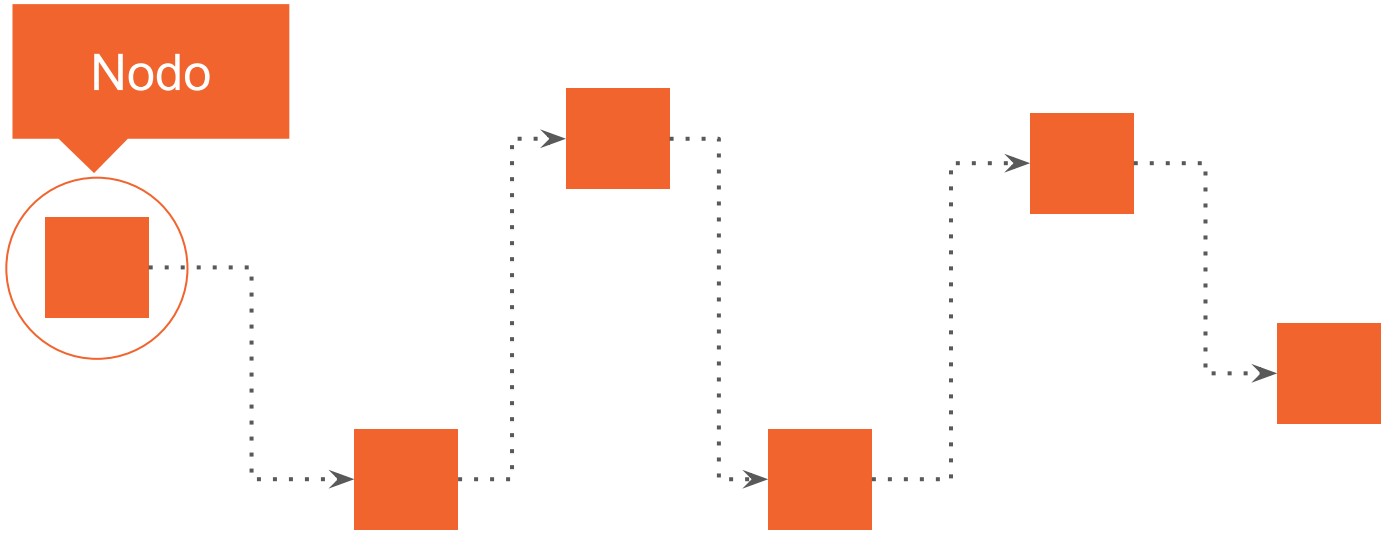
```
miAuto.m_iNumeroDeModelo = 2;

miAuto.m_iKilometraje = 30000;

miAuto.m_iCilindrada = 1800;
```

Listas enlazadas

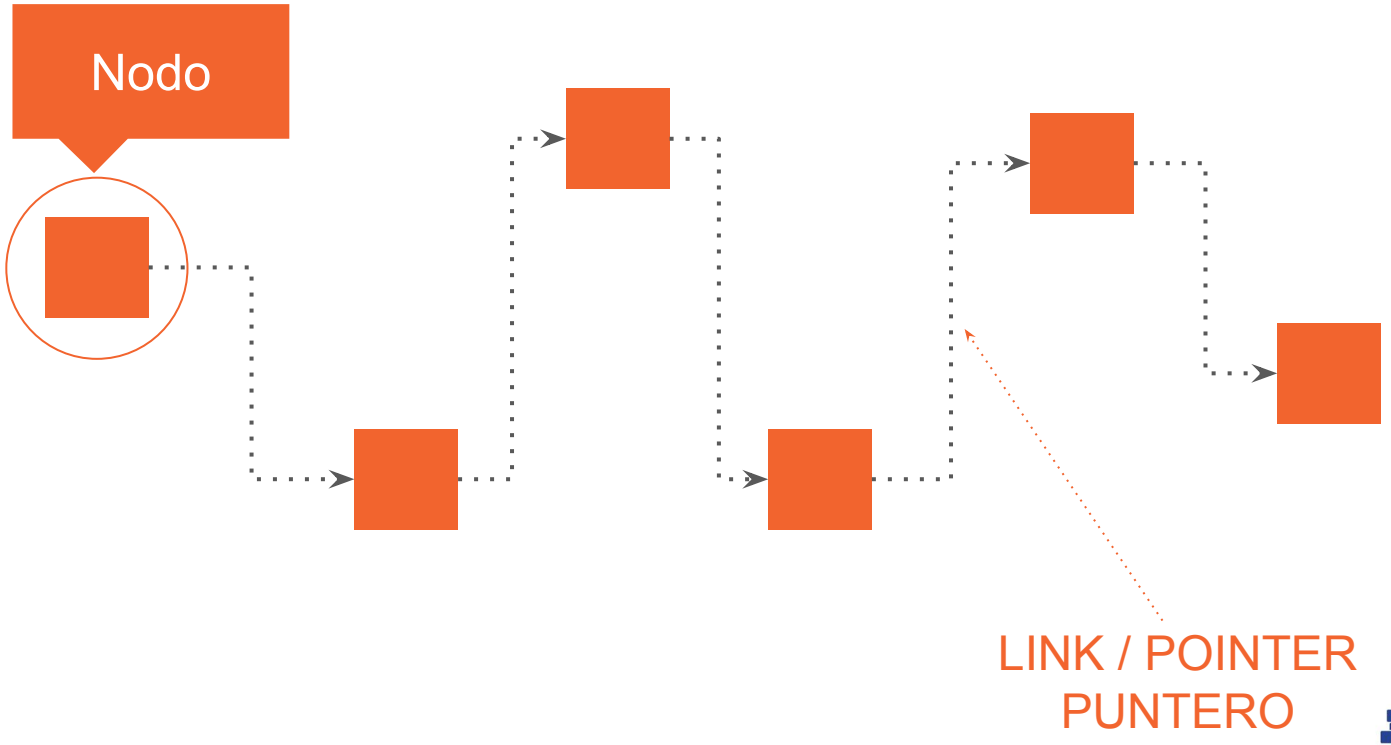
Listas enlazadas



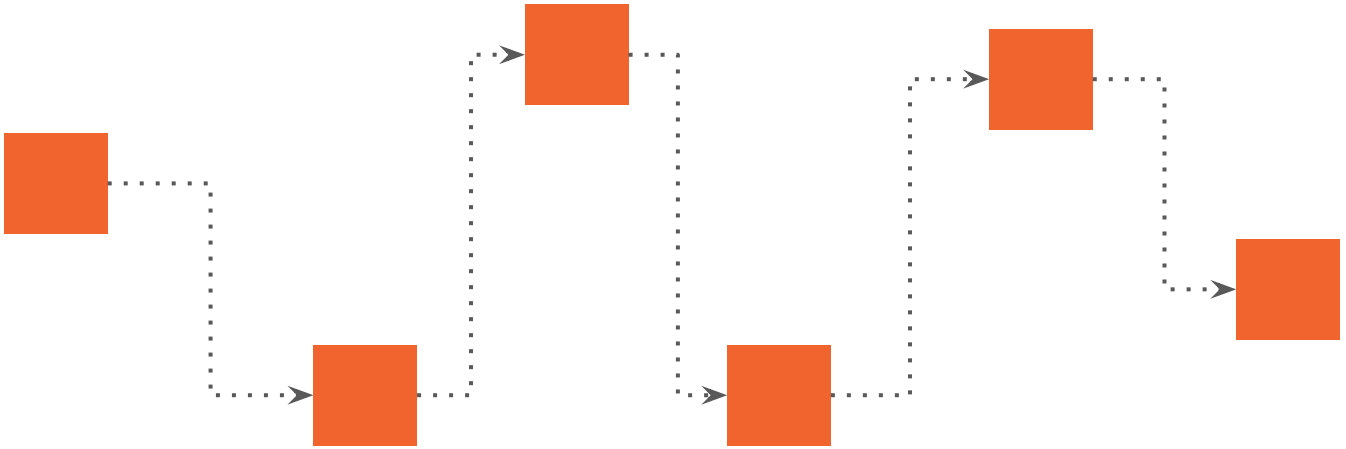
Listas enlazadas



Listas enlazadas



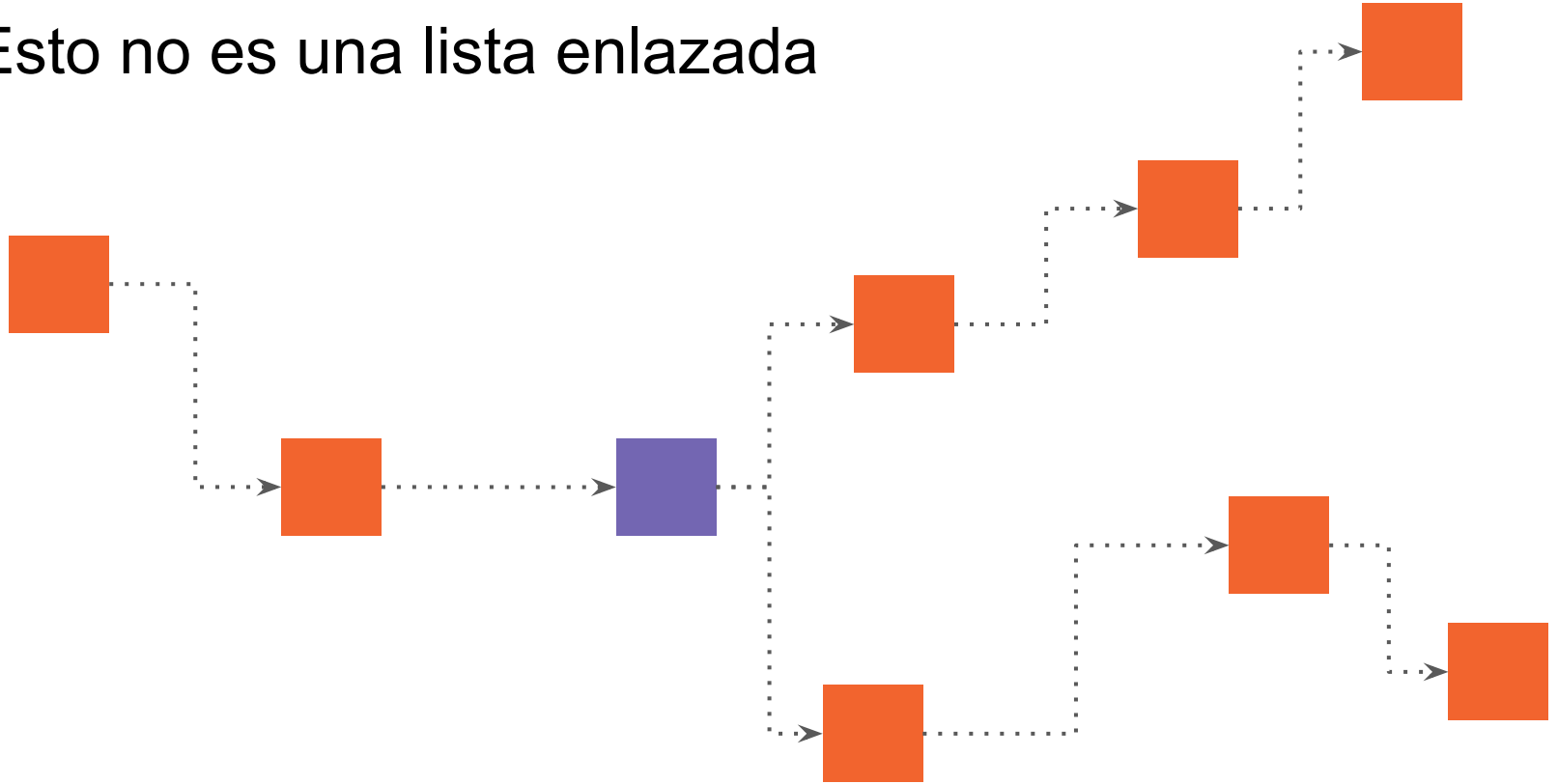
Listas enlazadas



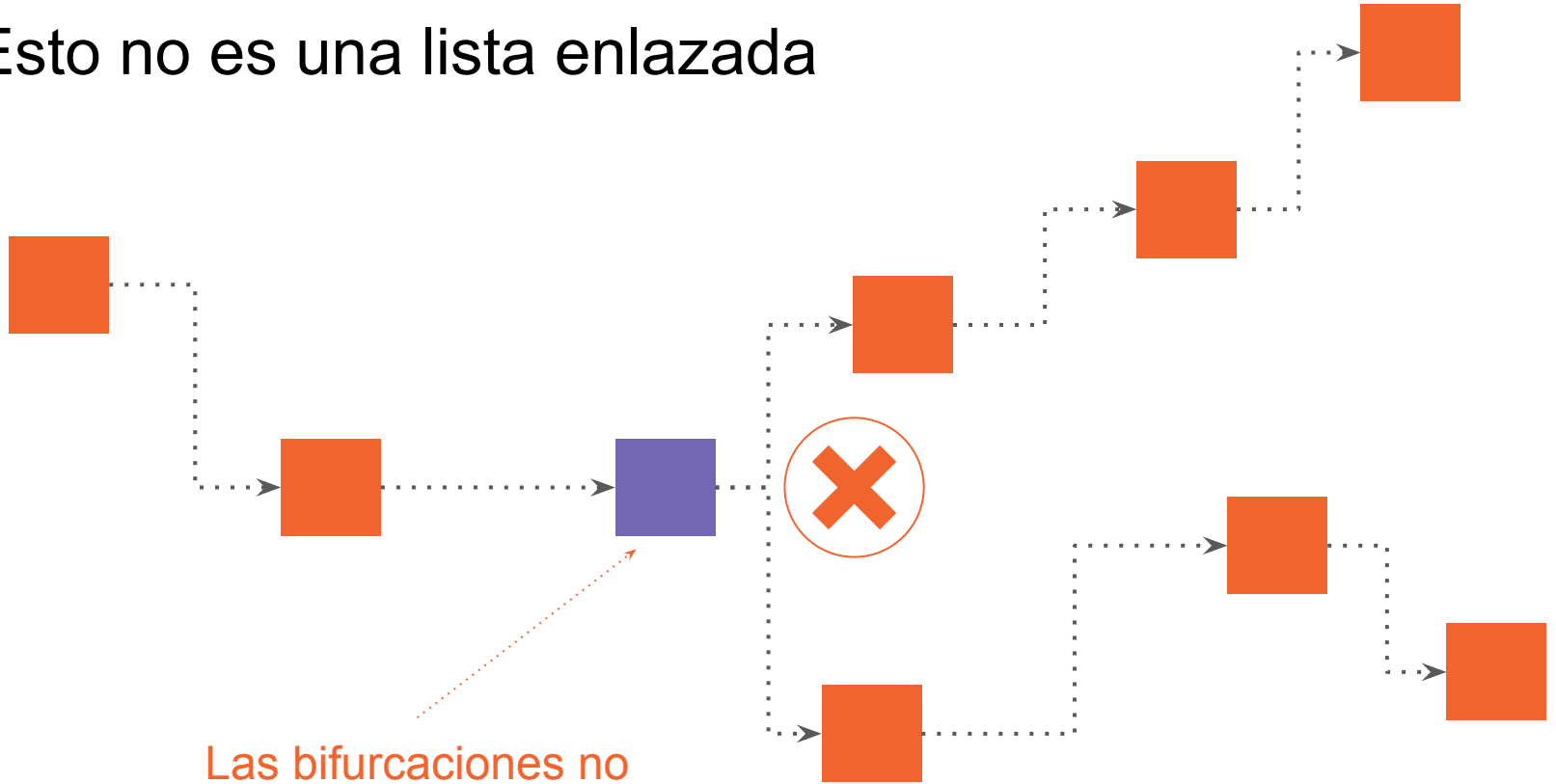
Secuencia lineal de nodos

sin bifurcaciones o ramas

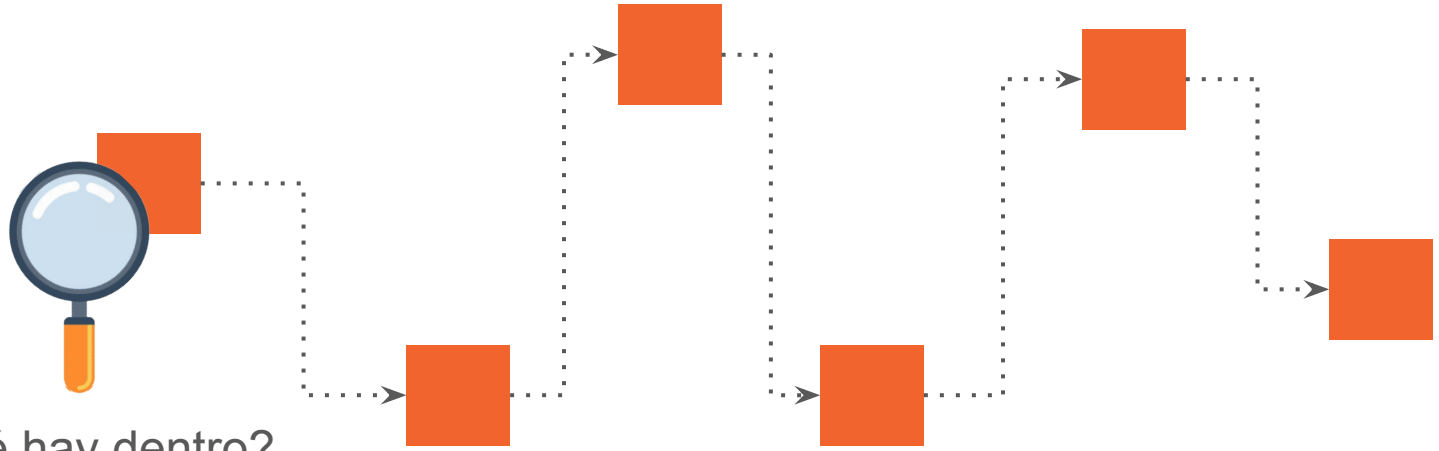
Esto no es una lista enlazada



Esto no es una lista enlazada

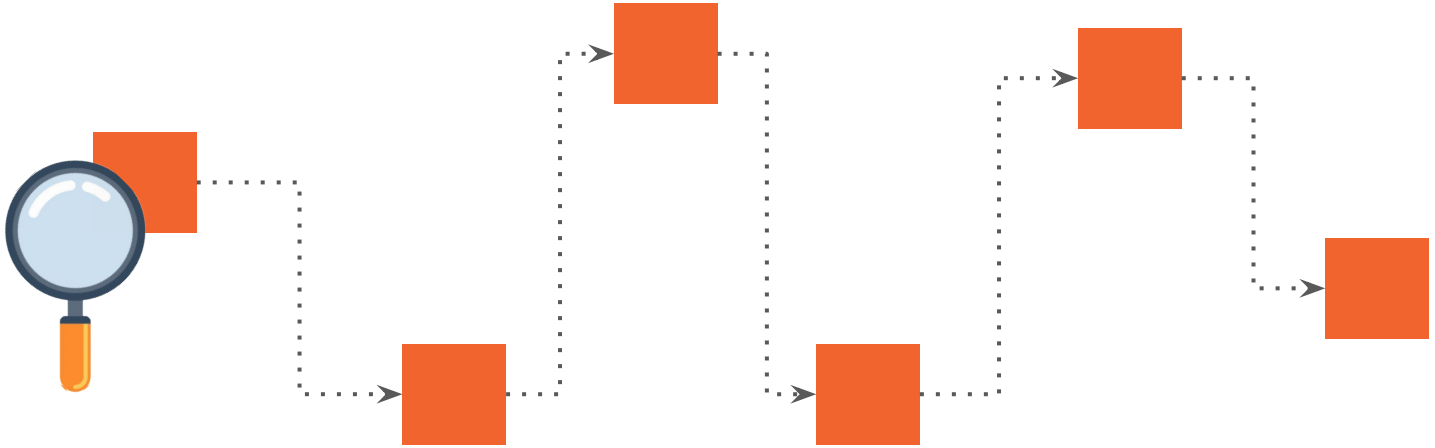


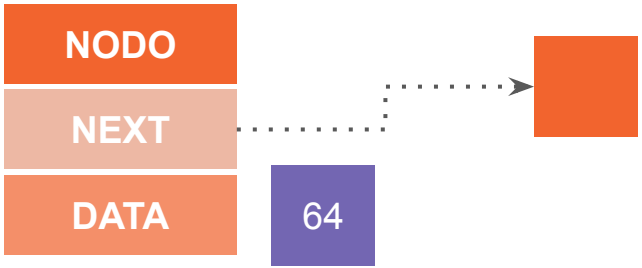
Que hay dentro de cada nodo en una lista enlazada?



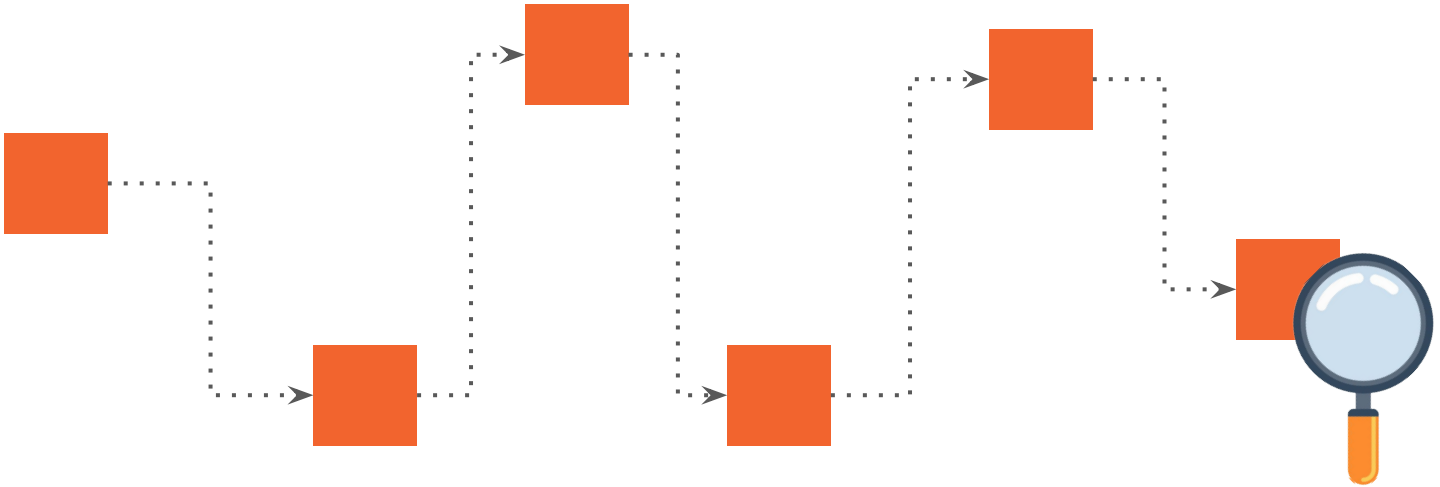
¿Qué hay dentro?

Que hay dentro de cada nodo en una lista enlazada?

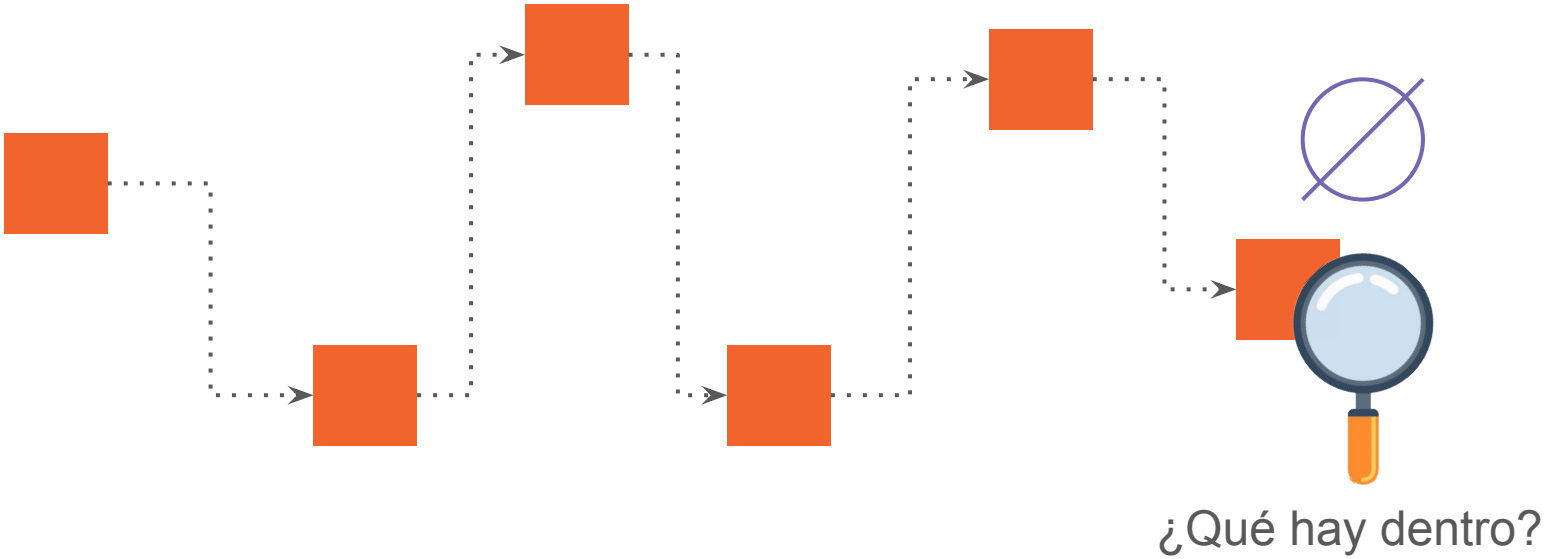




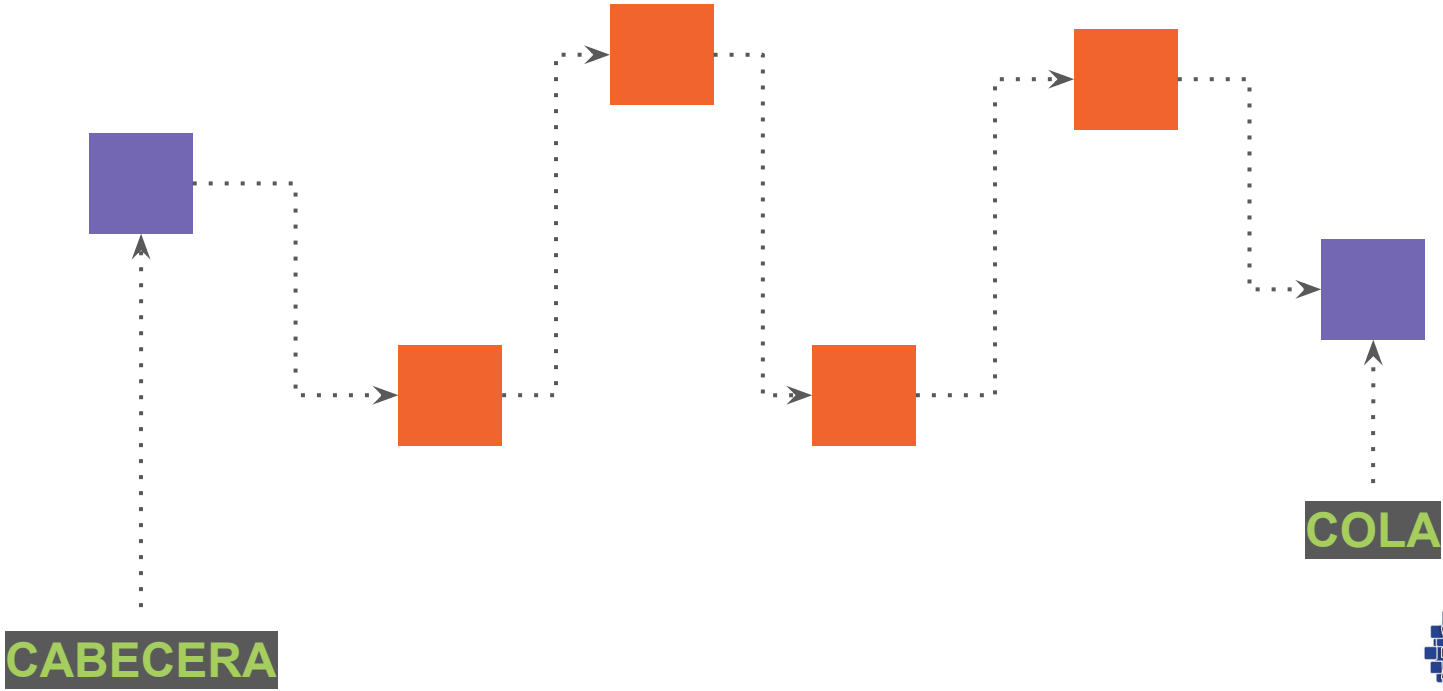
Que hay dentro de cada nodo en una lista enlazada?



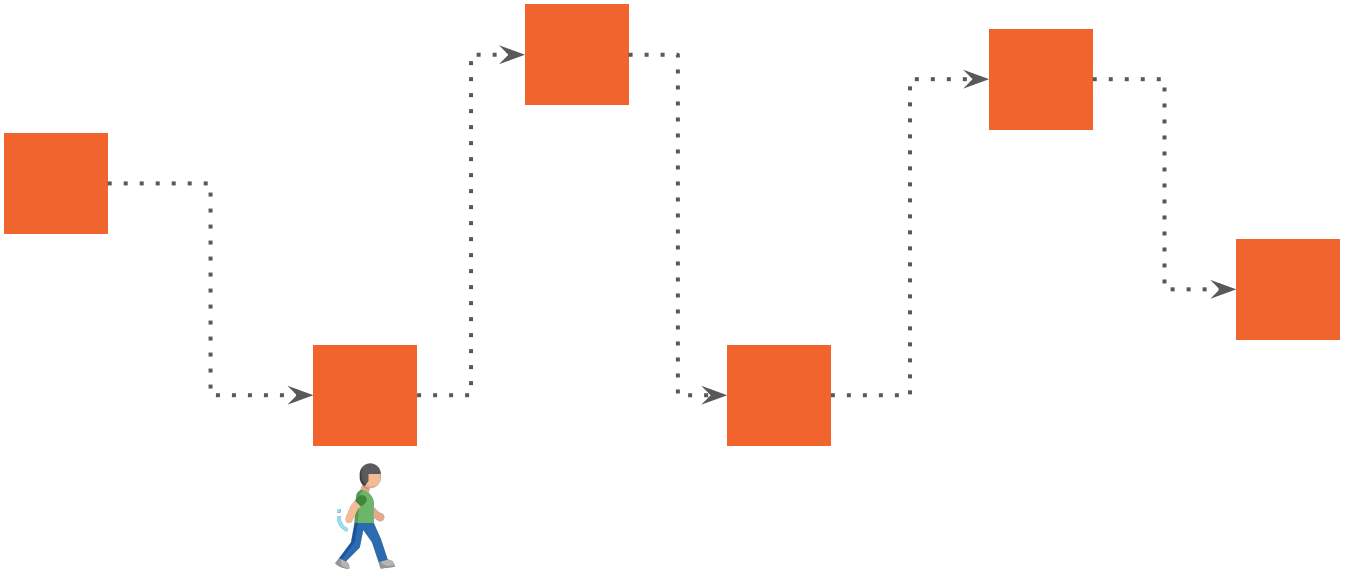
Que hay dentro de cada nodo en una lista enlazada?



Cabeza y cola de listas enlazadas



Recorriendo la lista enlazada simple



Listas enlazadas simples - Nuevo nodo

Crear nuevo nodo

- Asignar memoria y valor.

¿Lista vacía?

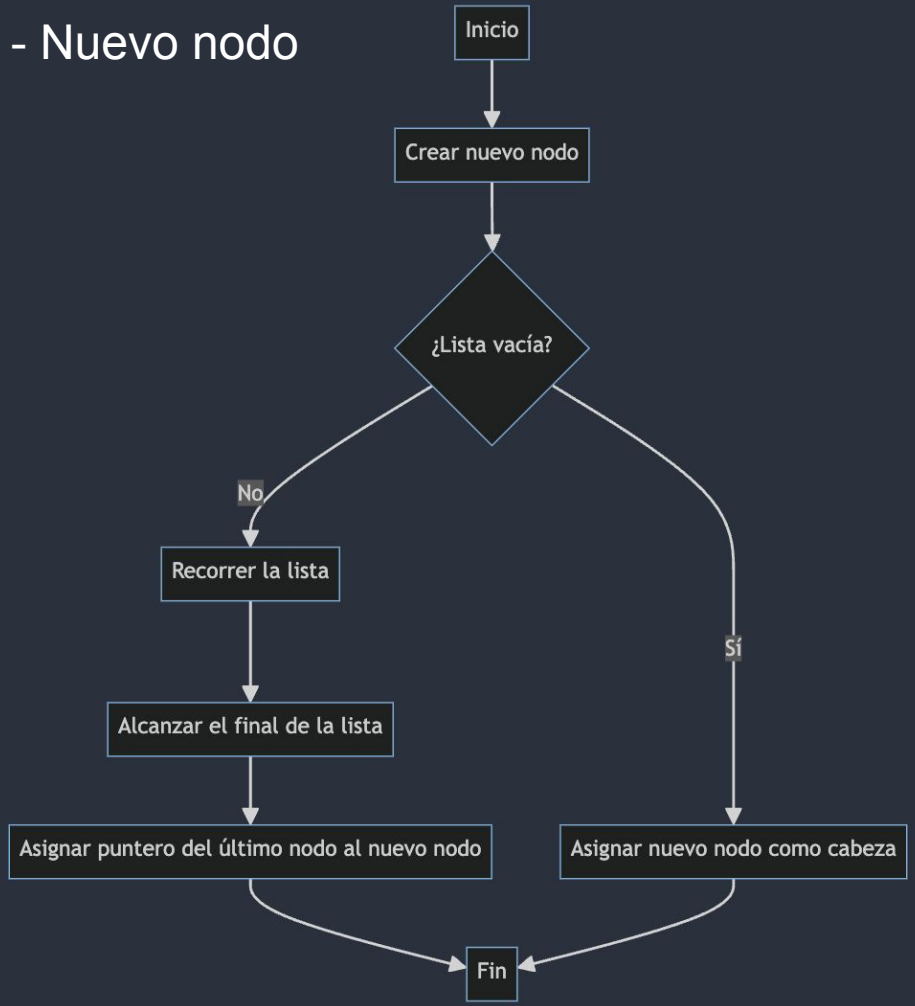
- **Sí**: Asignar nuevo nodo como cabeza.
- **No**: Continuar.

Recorrer la lista

- Usar puntero temporal para avanzar.

Alcanzar el final de la lista

- Asignar puntero del último nodo al nuevo nodo.



UTN San Nicolás
SEC. POSGRADO Y EDUCACIÓN CONTINUA

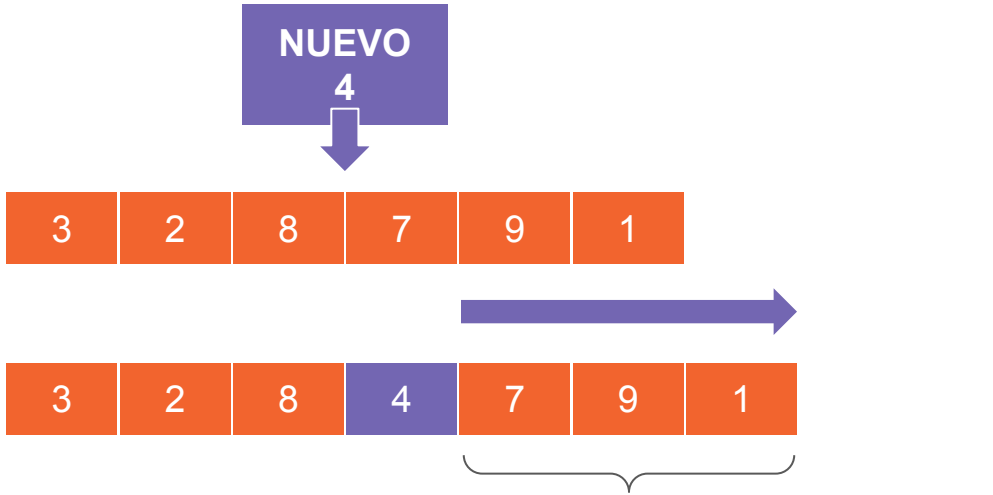
```
#include <iostream>
#include <cstdlib>
#include <cstring>

struct Nodo {
    char *Nombre;
    Nodo *sig;
};

Nodo *Lista = nullptr;

bool ListaVacia();
char *SacaLista();
void MuestraListado();
void InsertarLista(const char *Nom);

bool ListaVacia() {
    return Lista == nullptr;
}
```

```
char *SacaLista() {
    if (ListaVacia()) {
        printf("La lista está vacía.\n");
        return nullptr;
    }

    Nodo *temp = Lista;
    char *nombre = strdup(Lista->Nombre); // Copiamos el nombre
    Lista = Lista->sig;
    free(temp->Nombre); // Liberar la memoria del nombre del nodo
    free(temp); // Liberar la memoria del nodo

    return nombre;
}

void MuestraListado() {
    if (ListaVacia()) {
        printf("La lista está vacía.\n");
        return;
    }

    Nodo *temp = Lista;
    while (temp != nullptr) {
        printf("%s", temp->Nombre);
        temp = temp->sig;
    }
}
```

```
int main() {
    system("cls");

    // Ejemplo de inserción de nombres
    InsertarLista("Ana");
    InsertarLista("Juan");
    InsertarLista("Maria");

    MuestraListado();

    // Ejemplo de extracción de un elemento
    char *nombre = SacaLista();
    if (nombre) {
        printf("Elemento extraido: %s\n", nombre);
        free(nombre); // Liberar la memoria del nombre
extraído para no volver a encontrarme un valor en caso de
que se asigne el mismo espacio de memoria
    }

    MuestraListado();

    return 0;
}
```

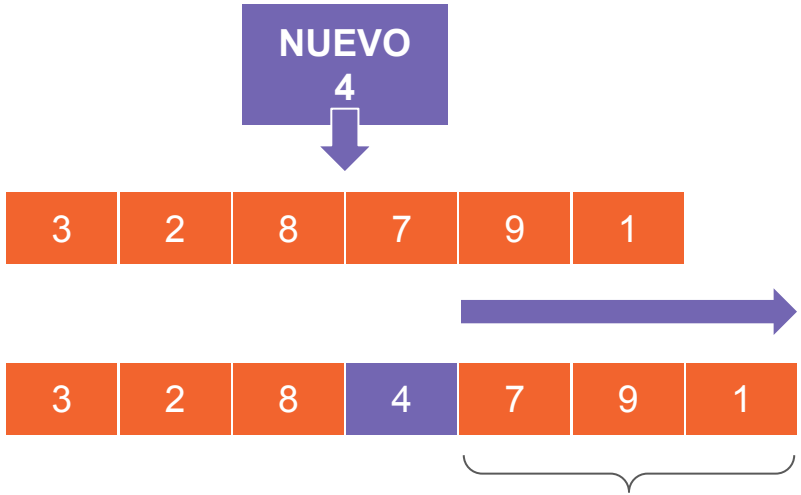
```
void InsertarLista(const char *Nom) {
    Nodo *t = new Nodo;
    if (!t) {
        printf("No se pudo asignar memoria.\n");
        return;
    }

    t->Nombre = strdup(Nom); // Asignamos el nombre
    t->sig = nullptr;

    if (Lista == nullptr) {
        Lista = t;
    } else {
        Nodo *q = Lista;
        while (q->sig != nullptr) {
            q = q->sig;
        }
        q->sig = t;
    }
}
```

Arrays VS Listas enlazadas: Insertando un elemento

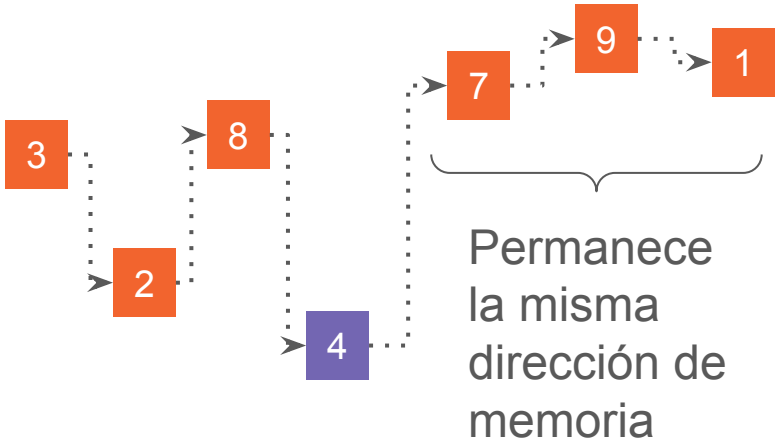


Cambia la dirección de memoria

Arrays VS Listas enlazadas: Insertando un elemento



Cambia la dirección de memoria



Permanece la misma dirección de memoria

No hay relocación

Tampoco hay sobrecarga de copia

UTN San Nicolás
SEC. POSGRADO Y EDUCACIÓN CONTINUA

Insertar nodo en una ubicación específica

Verificar si la lista está vacía o la posición es 0

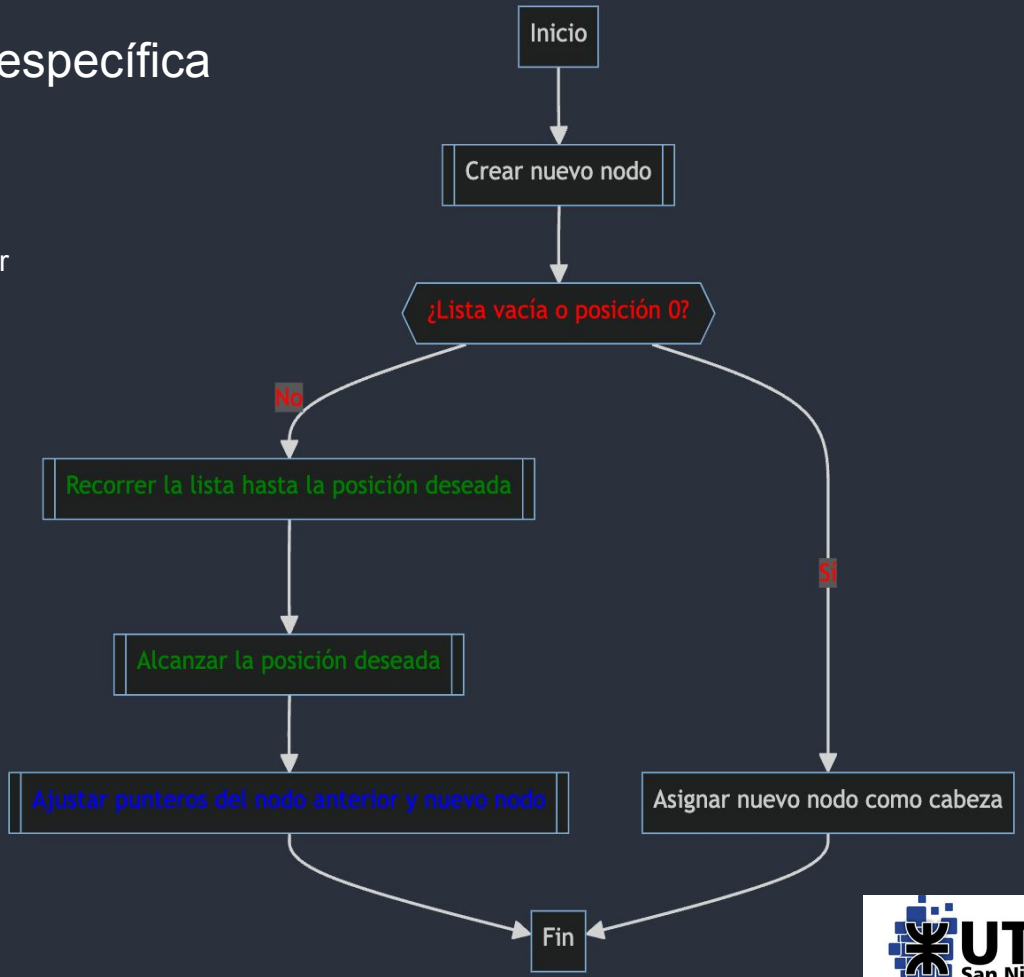
- Si está vacía o la posición es 0:
  - Asignar el nuevo nodo como el primer nodo (cabeza) de la lista.
- Sí no:
  - Continuar al siguiente paso.

Recorrer la lista hasta la posición deseada

- Usar un puntero temporal para avanzar a través de los nodos.
- Mantener un puntero al nodo anterior.
- Avanzar hasta llegar a la posición deseada.

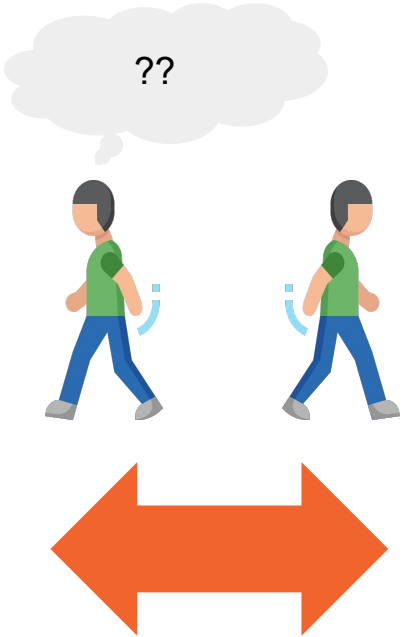
Insertar el nuevo nodo en la posición

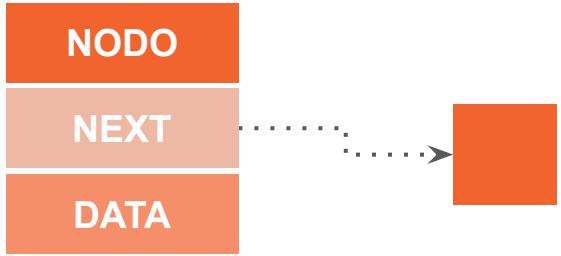
- Ajustar el puntero del nodo anterior para que apunte al nuevo nodo.
- Ajustar el puntero del nuevo nodo para que apunte al nodo actual.



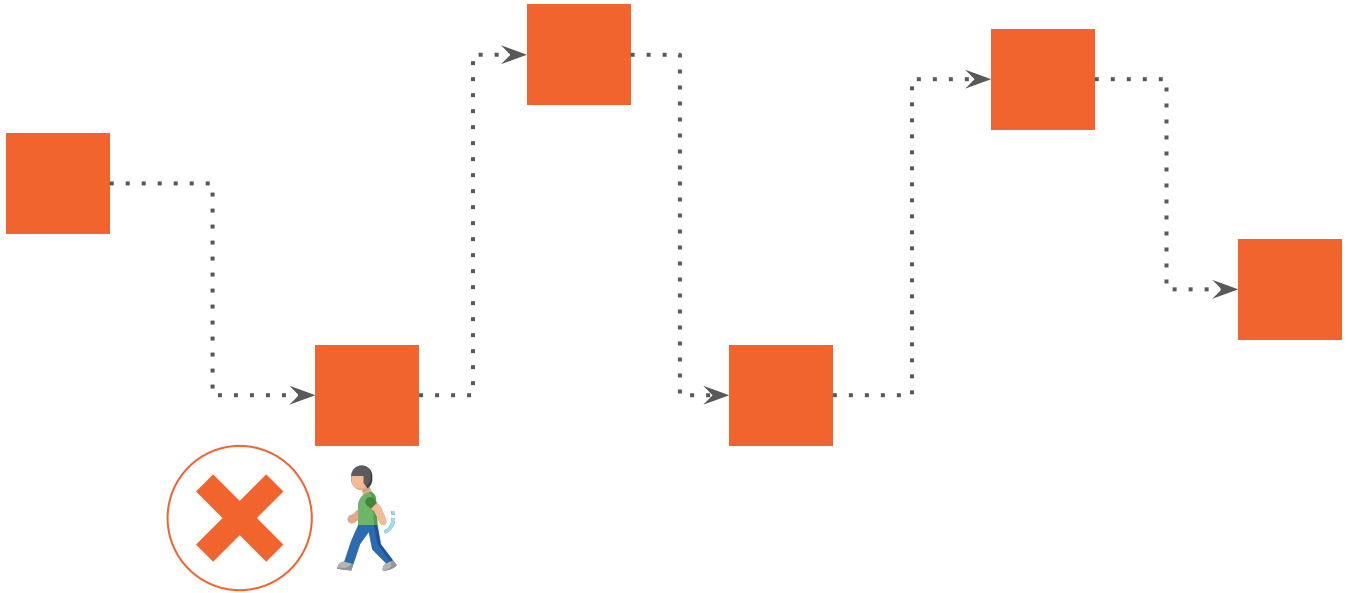
UTN San Nicolás
SEC. POSGRADO Y EDUCACIÓN CONTINUA

Cabeza y cola de listas enlazadas



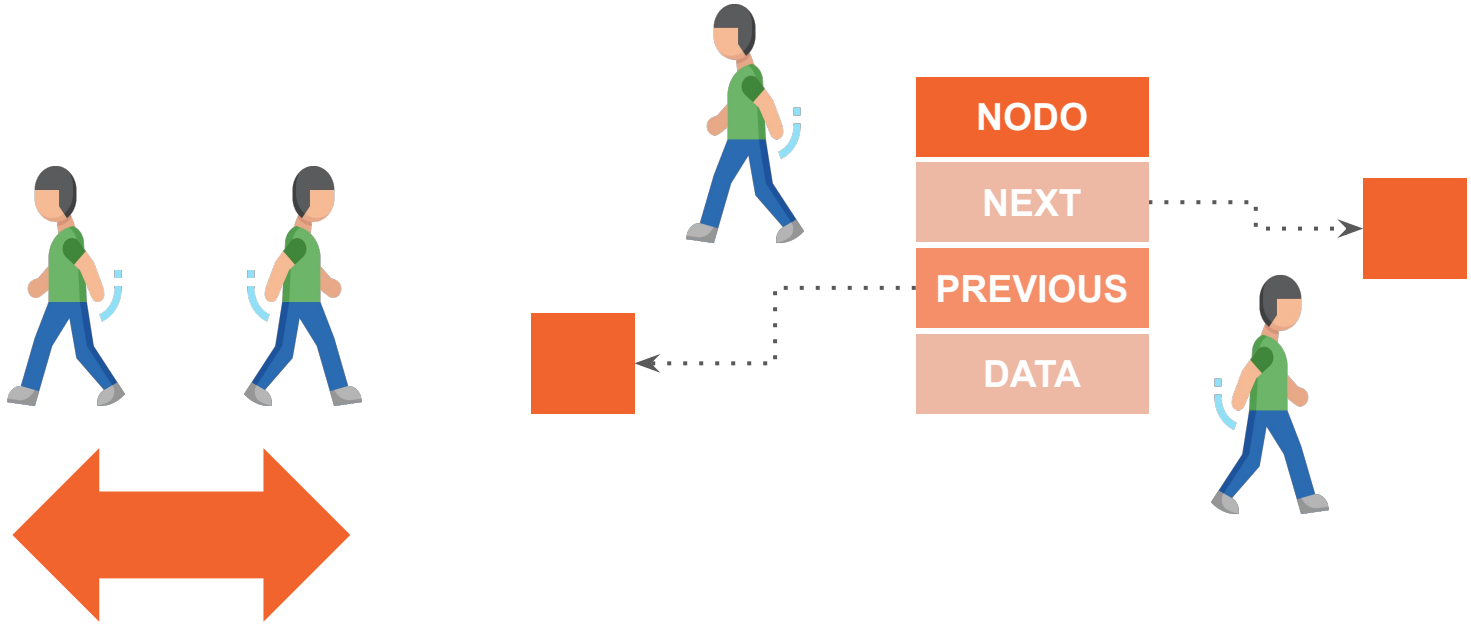


Cabeza y cola de listas enlazadas

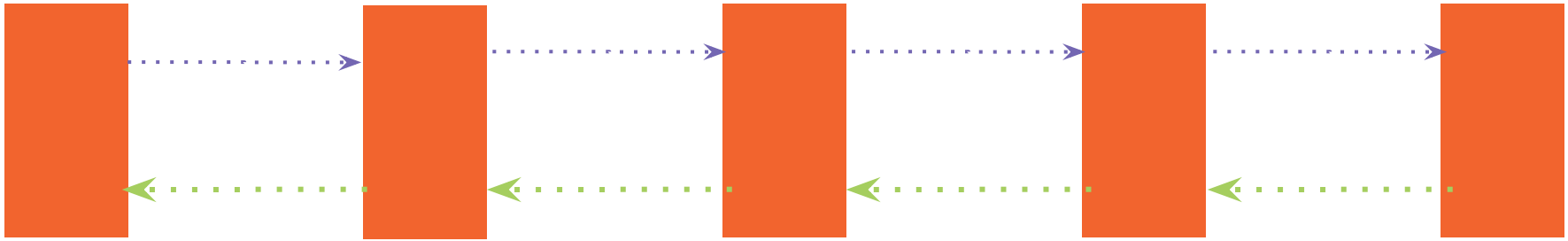


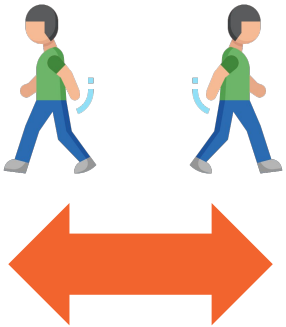
NO EXISTE; NODO -> PREVIOUS

Cabeza y cola de listas enlazadas



Listas doblemente enlazadas





Listas doblemente enlazadas

Crear nuevo nodo

- Reservar memoria para el nuevo nodo.
- Asignar valor al nuevo nodo.

Verificar si la lista está vacía

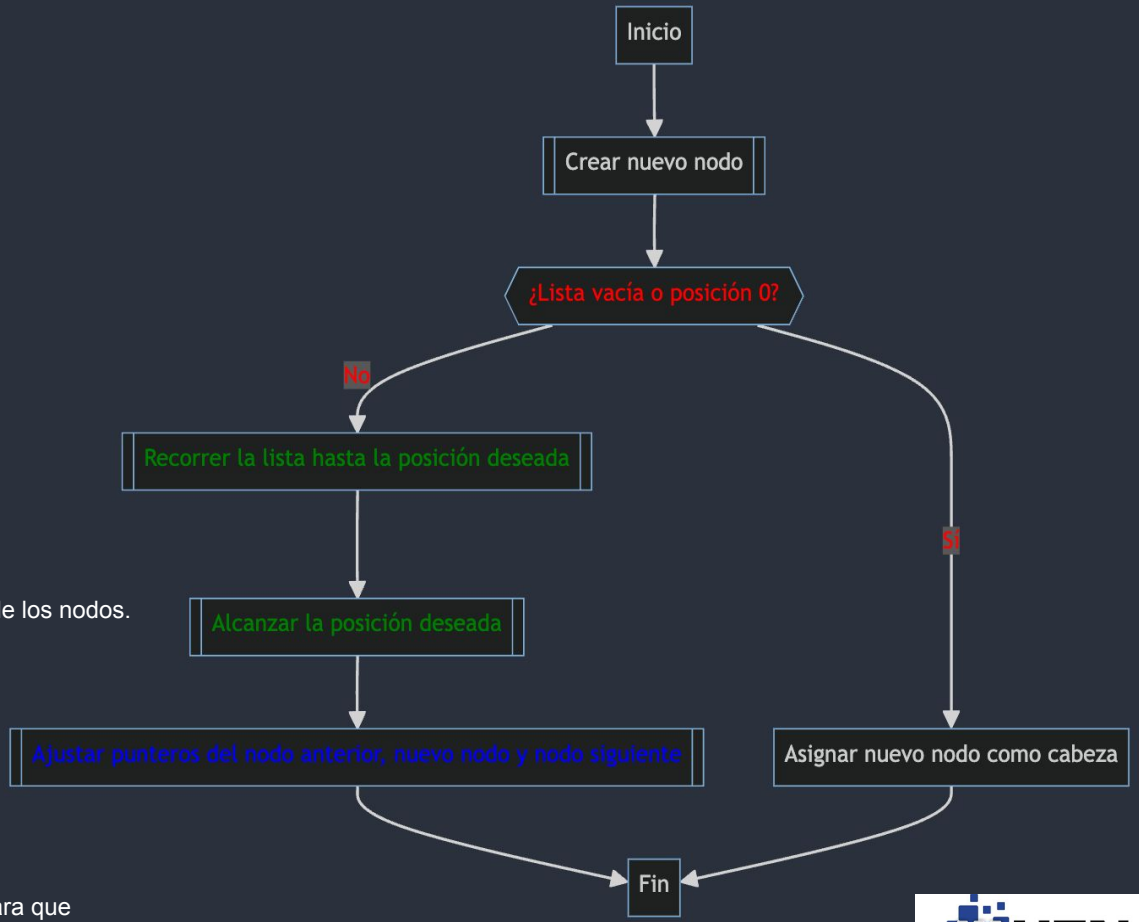
- Si está vacía:
  - Asignar el nuevo nodo como el primer nodo (cabeza) de la lista.
  - Terminar.
- Si no está vacía:
  - Continuar al siguiente paso.

Recorrer la lista hasta la posición deseada

- Usar un puntero temporal para avanzar a través de los nodos.
- Mantener un puntero al nodo anterior.
- Avanzar hasta llegar a la posición deseada.

Insertar el nuevo nodo en la posición

- Ajustar el puntero del nodo anterior para que apunte al nuevo nodo.
- Ajustar el puntero del nuevo nodo para que apunte al nodo actual.
- Ajustar el puntero del nodo siguiente (si existe) para que apunte al nuevo nodo.
- Ajustar el puntero del nuevo nodo para que apunte al nodo anterior.



UTN San Nicolás
SEC. POSGRADO Y EDUCACIÓN CONTINUA