Lists and Linked Lists

Lecture 7

1107186 – Estruturas de Dados

Prof. Christian Azambuja Pagot
CI / UFPB

Universidade Federal da Paraíba
Centro de Informática

What is a List?

- List is an **ADT**.
- A list represents a **finite ordered collection** of **values**.
- A certain value can appear **more than once** in a list.
- Operations:
  - **Prepending** an item to the list.
  - **Appending** an item to the list.
  - **Inserting** a item into the list.
  - **Deleting** an item.

How can we **implement** a list?

List Implementation

- Array-based.
- Linked list-based.

Arrays

- Are stored as **contiguous** chunks of memory.
- Have **fixed size**.
- Each **position** is referenced through an **index**.

The Costs of the Arrays

- Accessing an arbitrary element?
  - **The cost is constant!**
- **Example:**

C code excerpt:

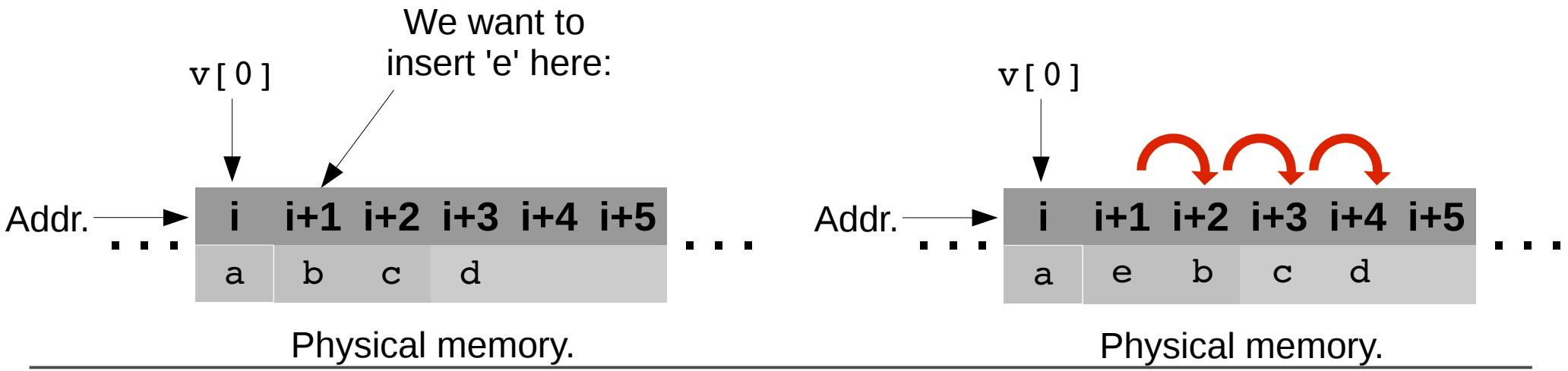
```
int v[10];

...

printf("%i", v[2]);

...
```

Jumps directly to the memory position where the 3rd array Item is stored.

The Costs of the Arrays

- Inserting a new element?
  - **The cost, in the worst case, may be *n* (where *n* is the length of the array)!**
- **Example:** **C code excerpt:**

```
char v[6]={'a','b','c','d','',''};
```



An Array-based List Implementation

C code excerpt:

```
struct ArrayList
{
    int* array;
    int size;
    int last;
};

...

struct ArrayList a;

a.size = 10;
a.array = (int*) malloc(a.size * sizeof(int));
a.last = -1;

Append(&a, 10);
```

Array list structure.

Array list declaration.

Array list initialization.

Insert element function.

Append(...)

C code excerpt:

```
void Append( struct ArrayList *list , int value ) {
    if ( list->last == ( list->size - 1 ) ) {
        list->size = ( !list->size ) ? 1 : list->size * 2;
        int* aux = ( int* ) malloc( sizeof( int ) * list->size );

        for ( int i = 0; i <= list->last; i++ )
            aux[i] = list->array[i];

        free( list->array );
        list->array = aux;
    }

    list->array[++list->last] = value;
}
```

Prepend(...)

C code excerpt:

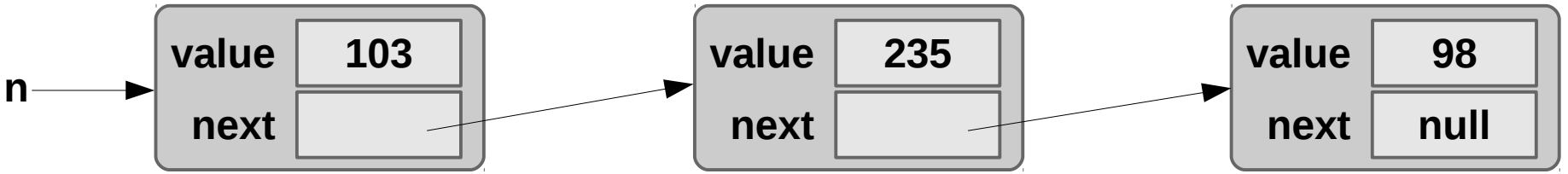
```
void Prepend( struct ArrayList *list , int value ) {
    if ( list->last < ( list->size - 1 ) )
        for ( int i = list->last; i >= 0; i-- )
            list->array[i+1] = list->array[i];
    else
        if ( !list->size ) {
            list->array = ( int* ) malloc( sizeof( int ) );
            list->size = 1;
        }
        else {
            int *aux;
            list->size *= 2;
            aux = ( int* ) malloc( sizeof( int ) * list->size );
            for ( int i = 0; i <= list->last; i++ )
                aux[i+1] = list->array[i];
            free( list->array );
            list->array = aux;
        }
    list->last++;
    list->array[0] = value;
}
```

Linked Lists

- Their **elements** (nodes) may be (and are likely to be) **spread** over the memory.
- **Nodes** are **connected** to each other through **pointers**.
- Lists have **varying sizes**.
- Each **position** is referenced through a **pointer**.

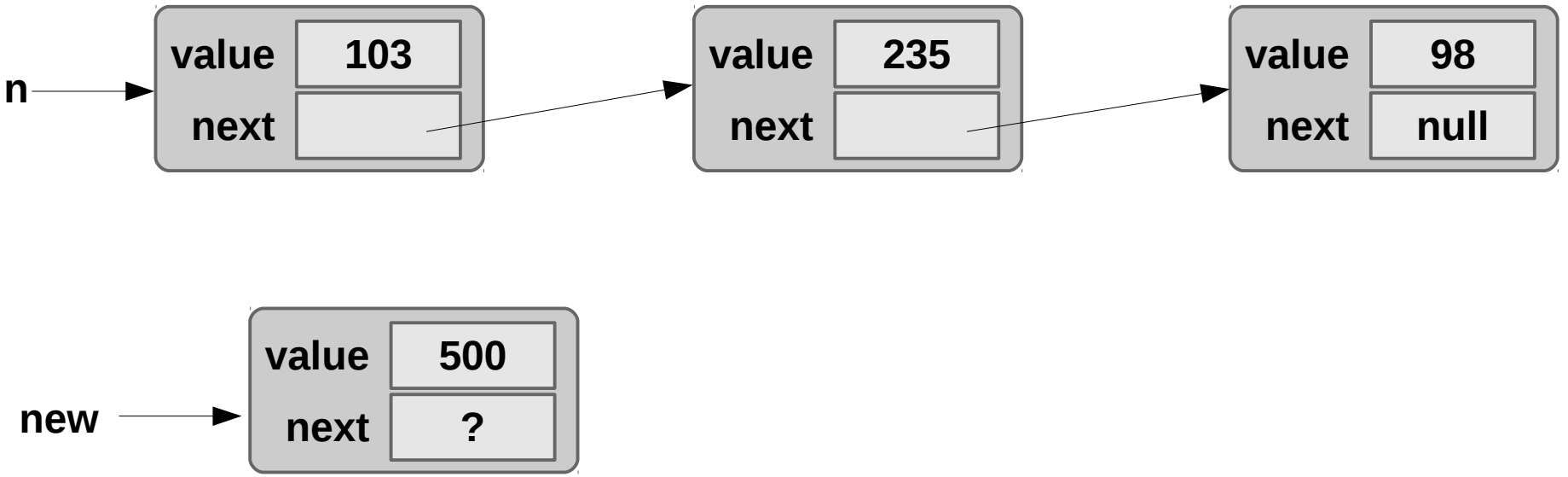
The Costs of Linked Lists

- Inserting a new element:
  - Given that we have the **pointer** of the **previous** node, **insertion is constant**.



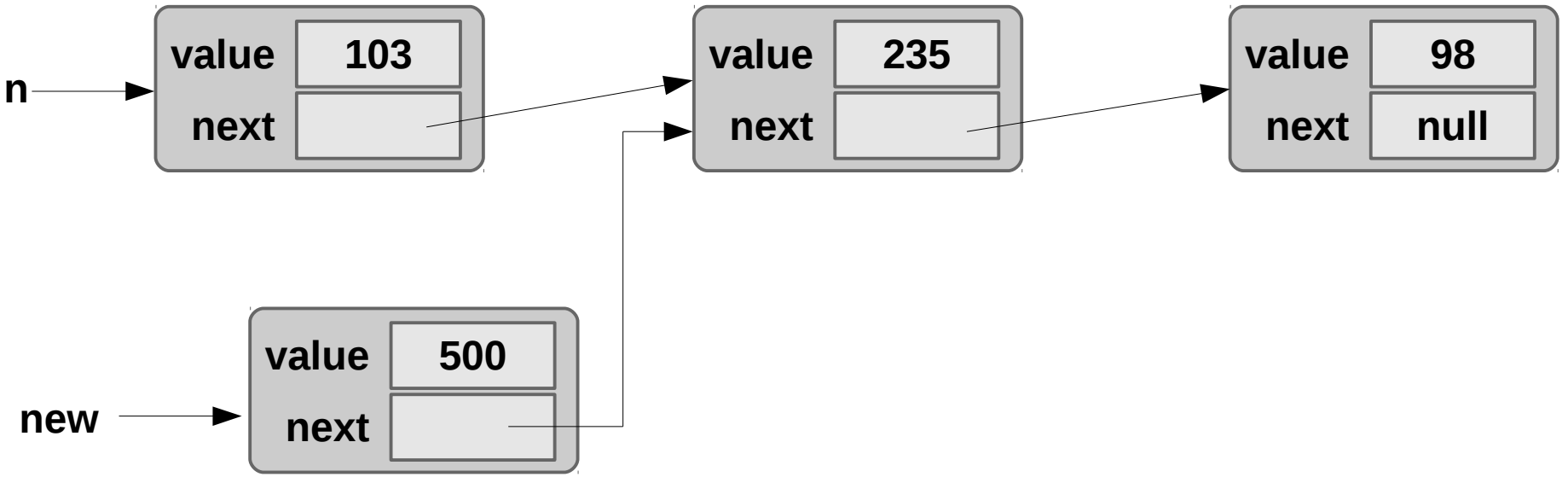
The Costs of Linked Lists

- Inserting a new element:
  - Given that we have the **pointer** of the **previous** node, **insertion is constant**.



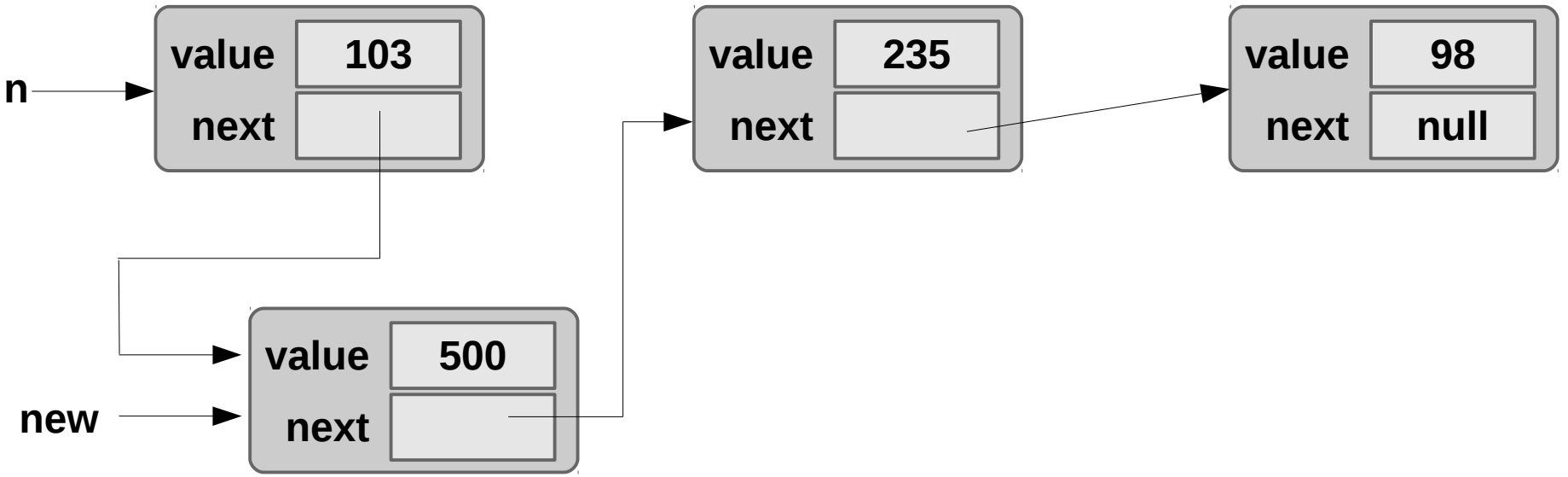
The Costs of Linked Lists

- Inserting a new element:
  - Given that we have the **pointer** of the **previous** node, **insertion is constant**.



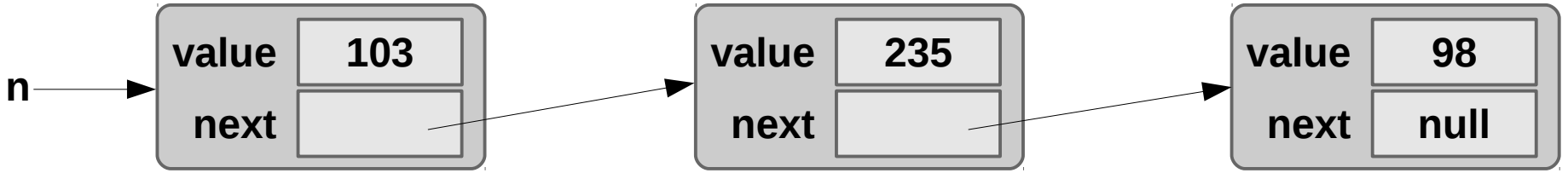
The Costs of Linked Lists

- Inserting a new element:
  - Given that we have the **pointer** of the **previous** node, **insertion is constant**.



The Costs of Linked Lists

- Accessing an arbitrary element:
  - The cost, in the **worst case**, is **n** (where n is the **length of the list**).
- **Example:**
  - Which is the value of the 3rd element?



A Linked List Implementation

C code excerpt:

```
struct Node
{
    int value;
    struct Node* next;
};

...

struct Node* n1;

n1 = (struct Node*) malloc (sizeof(struct Node));
n1->value = -1;
n1->next = NULL;

InsertAfter(n1, -2);
```

List node structure.

Pointer to a node.

Initialization of the 1st. node.

Insert element after node n1.

Append(...)

C code excerpt:

```
void Append( struct Node** node, int value ) {
    if ( !*node ) {
        *node = ( struct Node* ) malloc ( sizeof( struct Node ) );
        (*node)->value = value;
        (*node)->next = NULL;
    }
    else {
        struct Node *aux = *node;
        while ( aux->next )
            aux = aux->next;
        aux->next = ( struct Node* ) malloc ( sizeof( struct Node ) );
        aux->next->value = value;
        aux->next->next = NULL;
    }
}
```

Prepend(...)

C code excerpt:

```
void Prepend( struct Node** node, int value )
{
    if ( !*node ) {
        *node = ( struct Node* ) malloc ( sizeof( struct Node ) );
        (*node)->value = value;
        (*node)->next = NULL;
    }
    else {
        struct Node* new_node = (struct Node*) malloc (sizeof(struct Node));
        new_node->value = value;
        new_node->next = *node;
        *node = new_node;
    }
}
```

InsertAfter(...)

C code excerpt:

```
void InsertAfter(struct Node* n, int val)
{
    struct Node* new_n;

    new_n = (struct Node*) malloc (sizeof(struct Node));
    new_n->value = val;
    new_n->next = n->next;
    n->next = new_n;
}
```

An Issue Related to Our Implementation

C code excerpt:

```
int main(void)
{
    struct Node *n0 = NULL;
    struct Node *n1 = NULL;

    Append( &n0, 10 );
    Append( &n0, 20 );
    Append( &n0, 30 );

    n1 = n0;

    Prepend ( &n0, -10 );

    PrintList( n0 );
    PrintList( n1 );

    ...
```

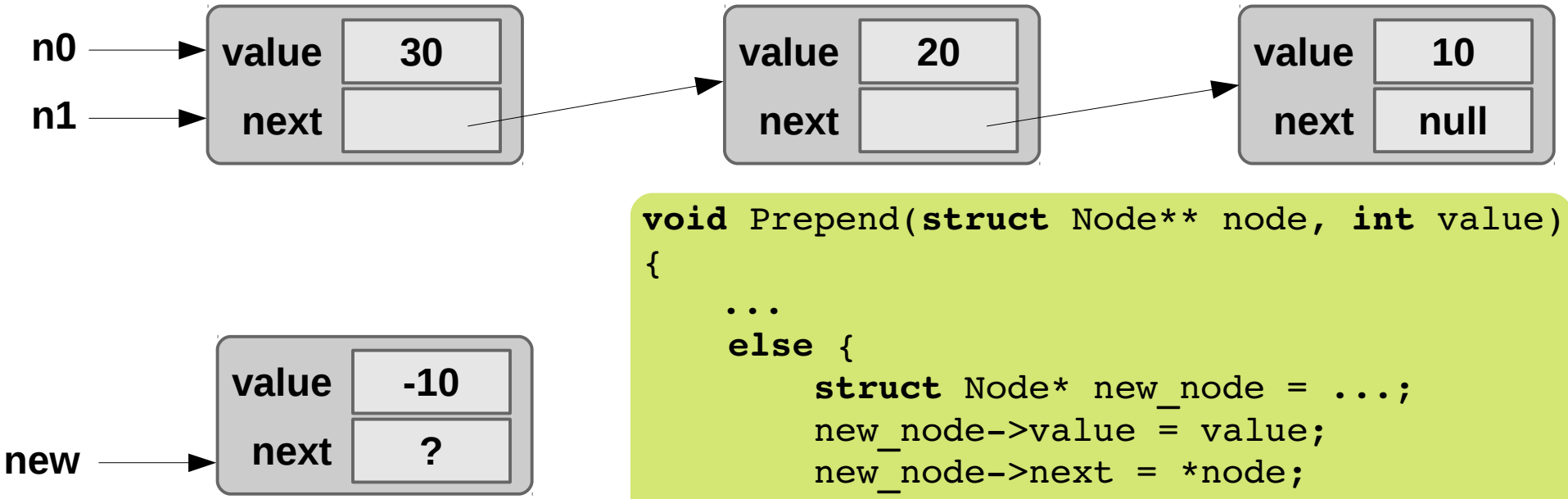
```
list[0] : -10
list[1] : 10
list[2] : 20
list[3] : 30
```

```
list[0] : 10
list[1] : 20
list[2] : 30
```

That's **bad!**

An Issue Related to Our Implementation

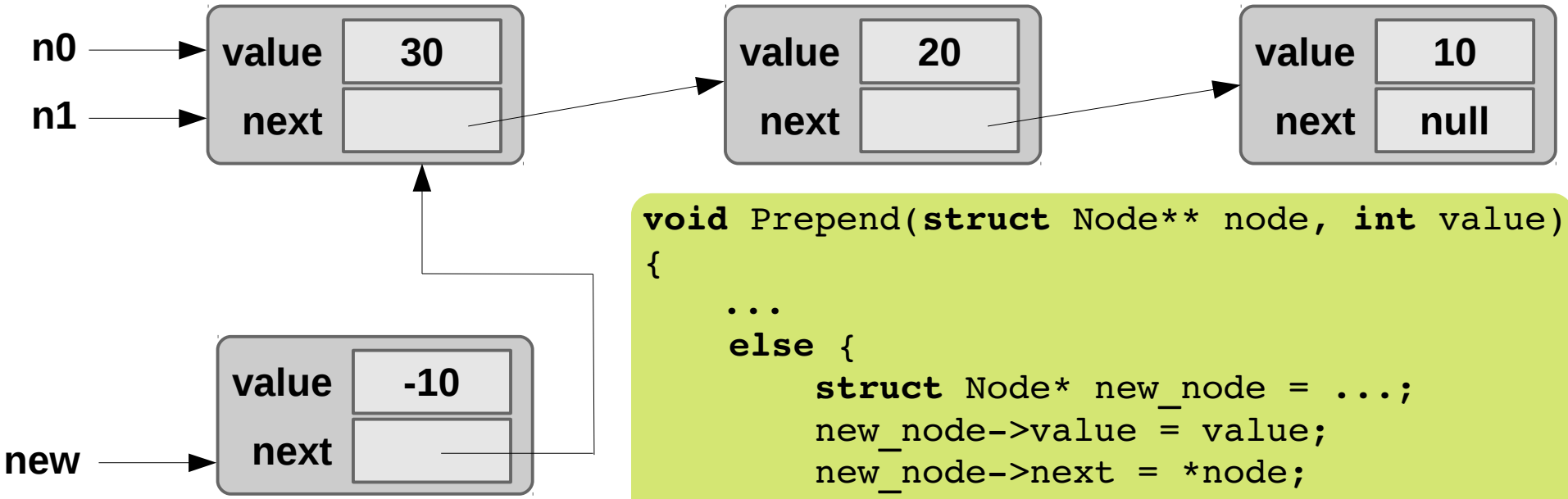
- What happened?




```
void Prepend(struct Node** node, int value)
{
    ...
    else {
        struct Node* new_node = ...;
        new_node->value = value;
        new_node->next = *node;
        *node = new_node;
    }
}
...
Prepend(&n0,-10);
```

An Issue Related to Our Implementation

- What happened?

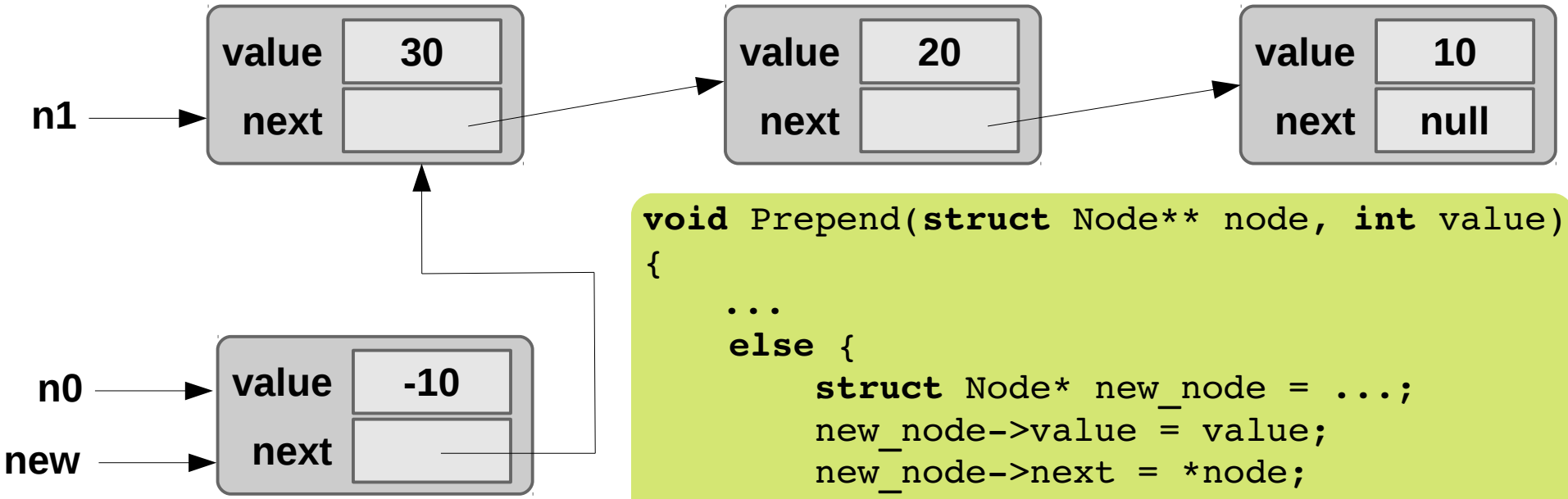



```
void Prepend(struct Node** node, int value)
{
    ...
    else {
        struct Node* new_node = ...;
        new_node->value = value;
        new_node->next = *node;
        *node = new_node;
    }
}
...
Prepend(&n0,-10);
```

An Issue Related to Our Implementation

- What happened?

n0 and n1 **do not point** to the **same node** anymore!




```
void Prepend(struct Node** node, int value)
{
    ...
    else {
        struct Node* new_node = ...;
        new_node->value = value;
        new_node->next = *node;
        *node = new_node;
    }
}
...
Prepend(&n0,-10);
```

One Possible Solution

- We create a **new struct** that contains a **pointer** to the **head** of the **linked list**:

C code excerpt:

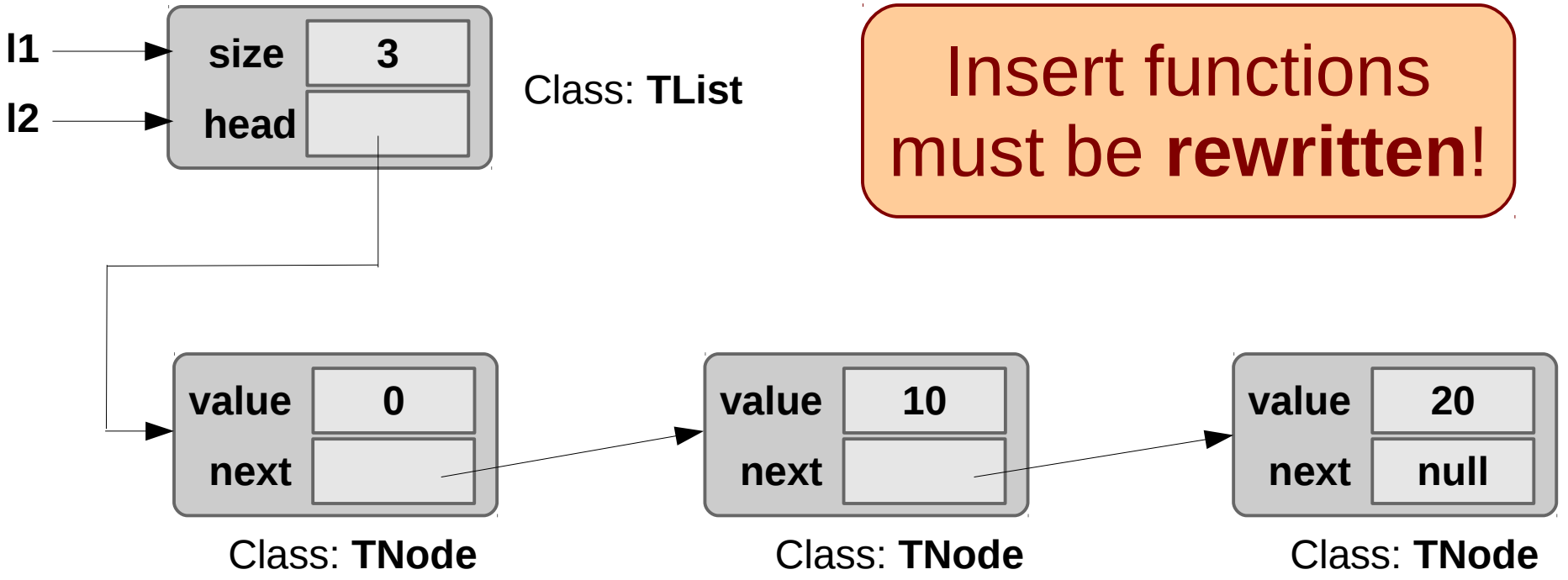
```
struct SLList
{
    struct Node* head;
    int size;
};
```

It can also contain important information about the list, such as the length.

```
struct Node
{
    int value;
    struct Node* next;
};
```

One Possible Solution

- We create a **new struct** that contains a **pointer** to the **head** of the **linked list**:



Discussion

- **Considering the presented implementations**:
  - Which is the 'best' approach: array-based or linked list based list?
  - How can we remove elements from the lists?

Think about it !!!