Listas, Pilas y Colas

Bernal Chauayo, Luis Antonio
Lacuaña Apaza, Margarita
Mendoza Villarroel, Alexis
Villena Zevallos, Ademir

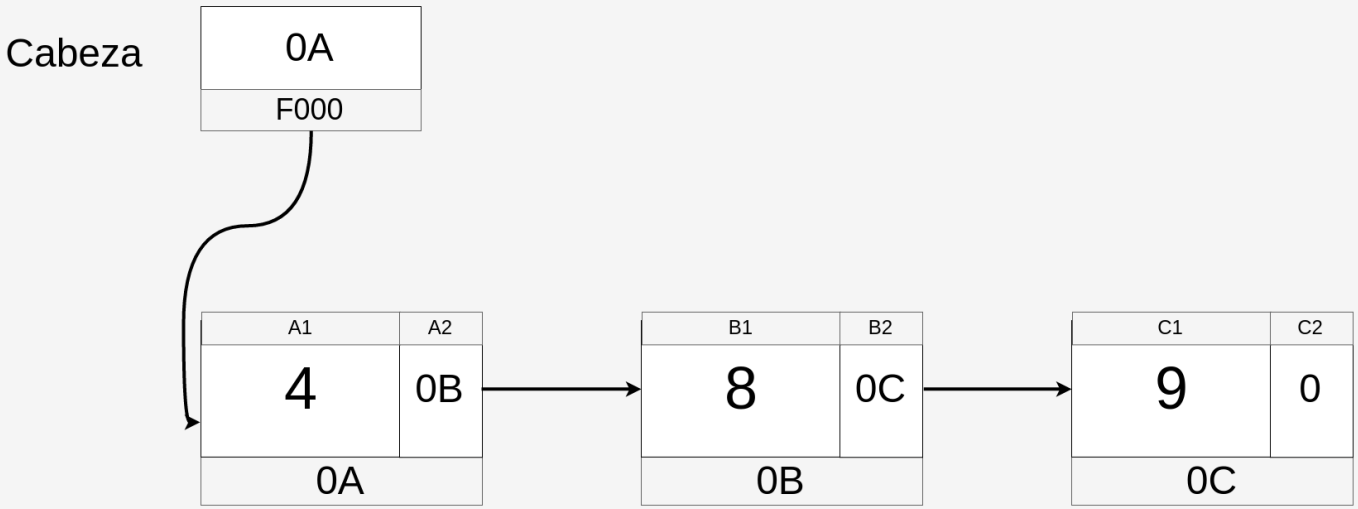
Ciencia de la Computación
Universidad Nacional de San Agustin

21 de Octubre del 2016

Índice

Lista simple

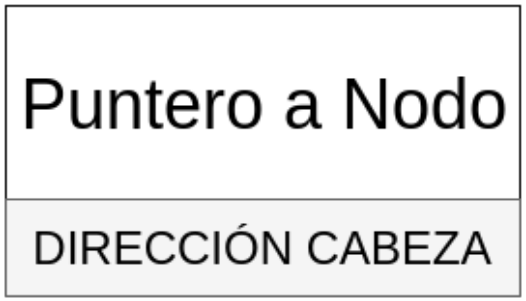
- Una lista enlazada simple es una estructura de datos formada por nodos, en la que cada nodo apunta al siguiente. De este modo guardando una referencia a la cabeza de la lista podemos acceder a todos los elementos de la misma.



- Nodo


DIRECCIÓN DATO
DIRECCIÓN PUNTERO
DATO
PUNTERO
a Nodo
DIRECCION NODO

- Lista


Puntero a Nodo
DIRECCIÓN CABEZA

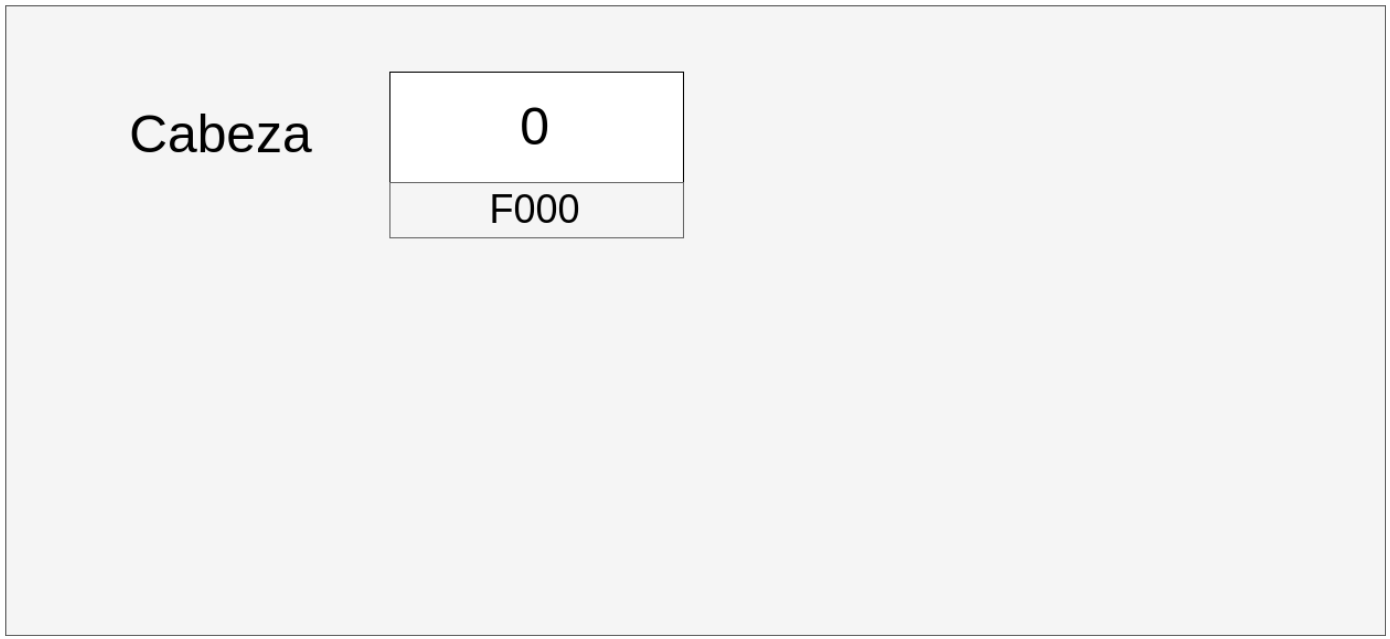
Nodo.

```
1   template<class T>
2   class Nodo
3   {
4   public:
5       ~Nodo(){}
6       Nodo(T d){  m_dato=d;  m_pSig=0; }
7   private:
8       Nodo<T>* m_pSig;
9       T m_dato;
10  friend class Lista<T>;
11  };
```

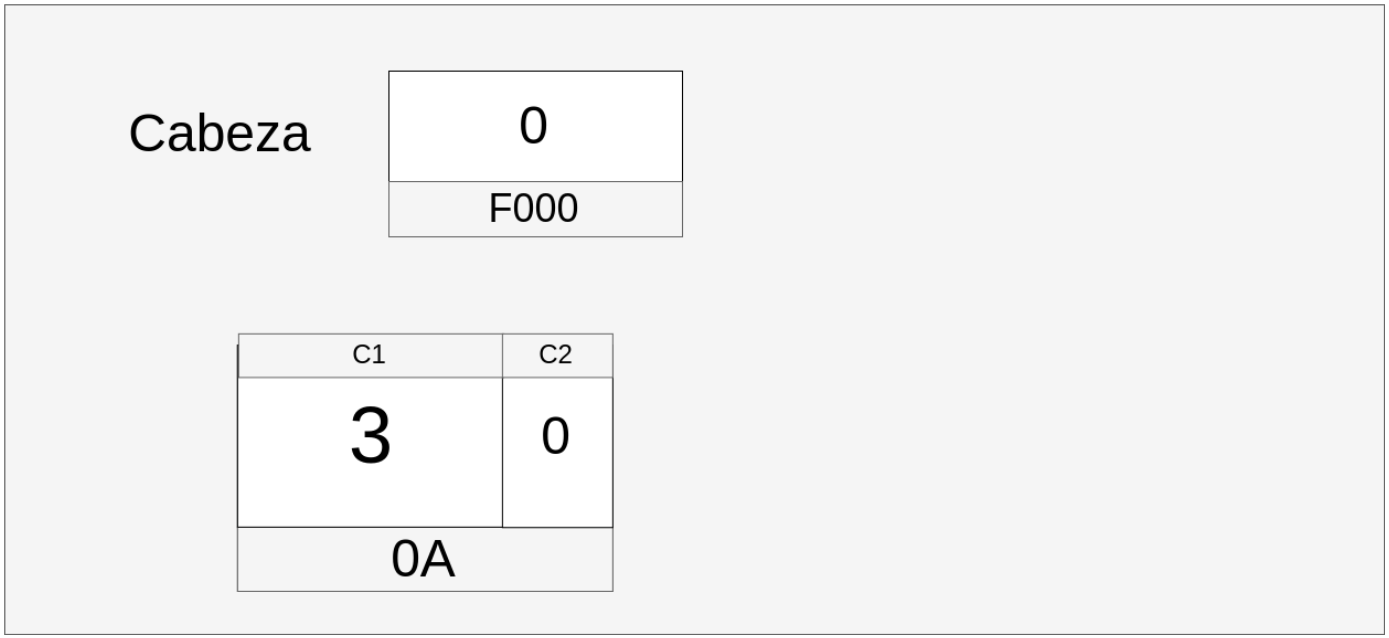
Lista.

```
1   template<class T>
2   class Lista
3   {
4   public:
5       Lista() { m_pHead=0; }
6       ~Lista(){}
7       bool find(T d, Nodo<T>**& p);
8       bool add(T d);
9       bool remove(T d);
10      void imprimir();
11  private:
12      Nodo<T>* m_pHead;
13  };
```

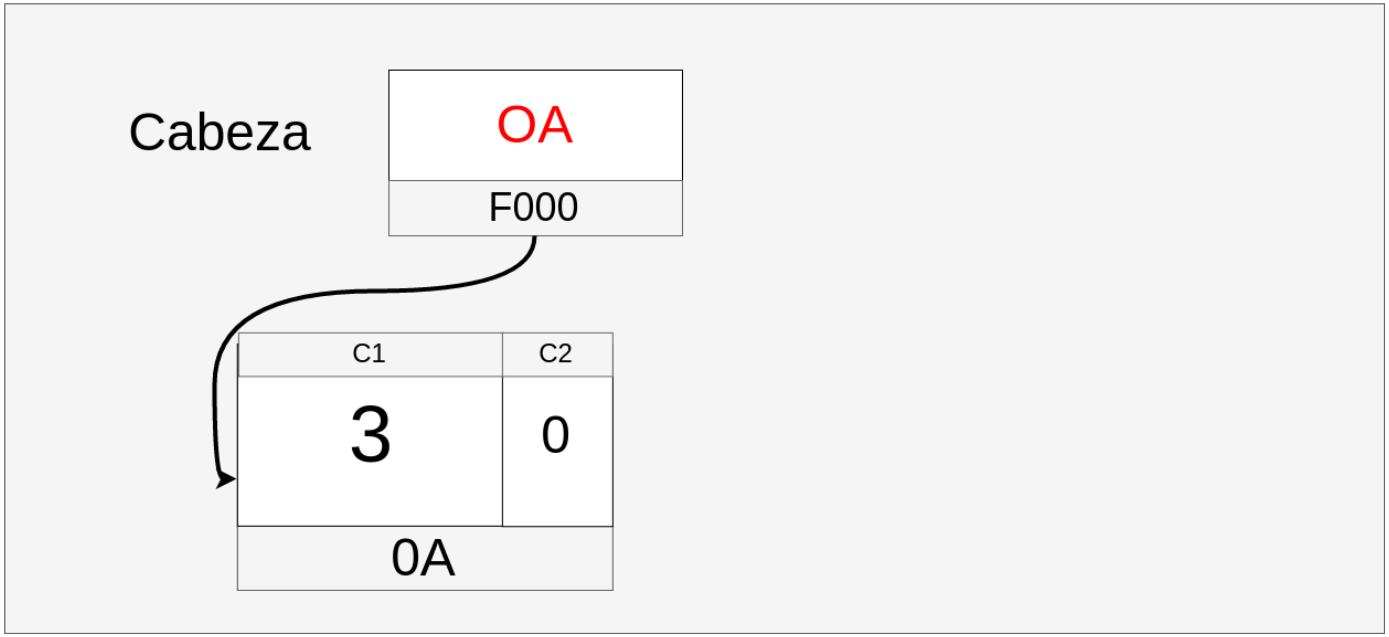
Inserción cuando el puntero a la cabeza es 0.



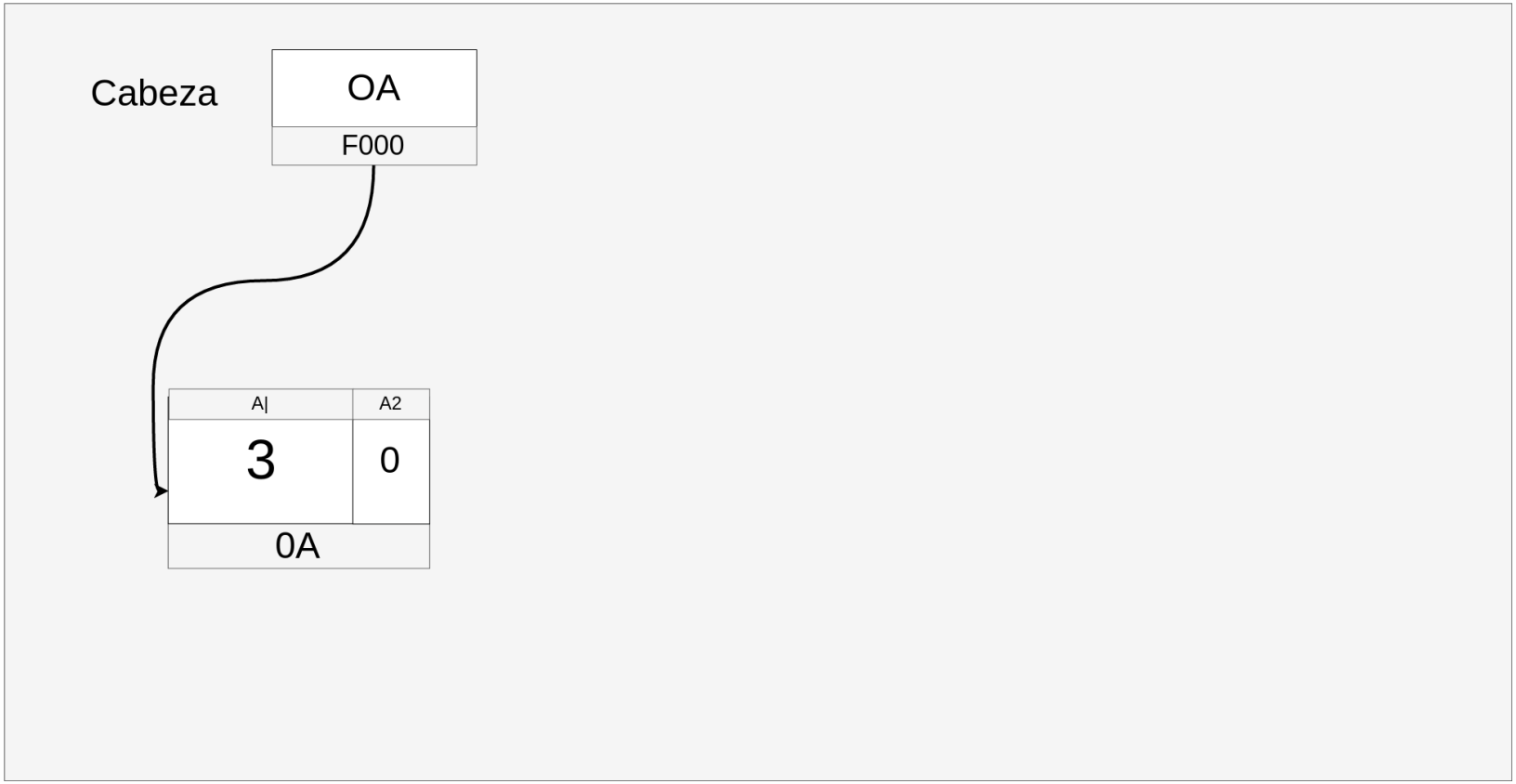
1.- Creamos un nuevo nodo con el dato a insertar (3).



2.- Hacemos que la cabeza apunte al nuevo nodo.

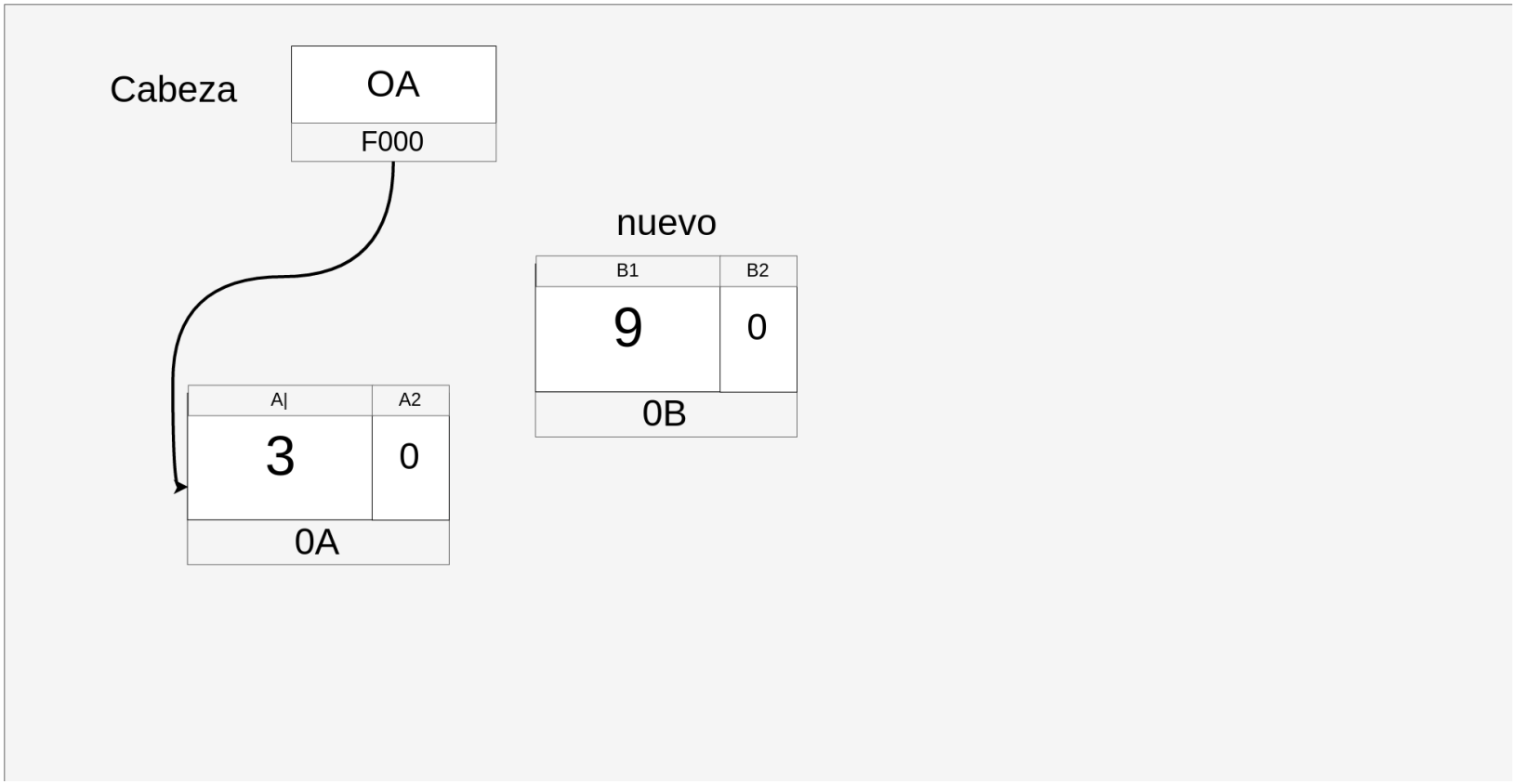


Inserción cuando el puntero a la cabeza es diferente de 0 y el dato a insertar es mayor que el dato de la cabeza

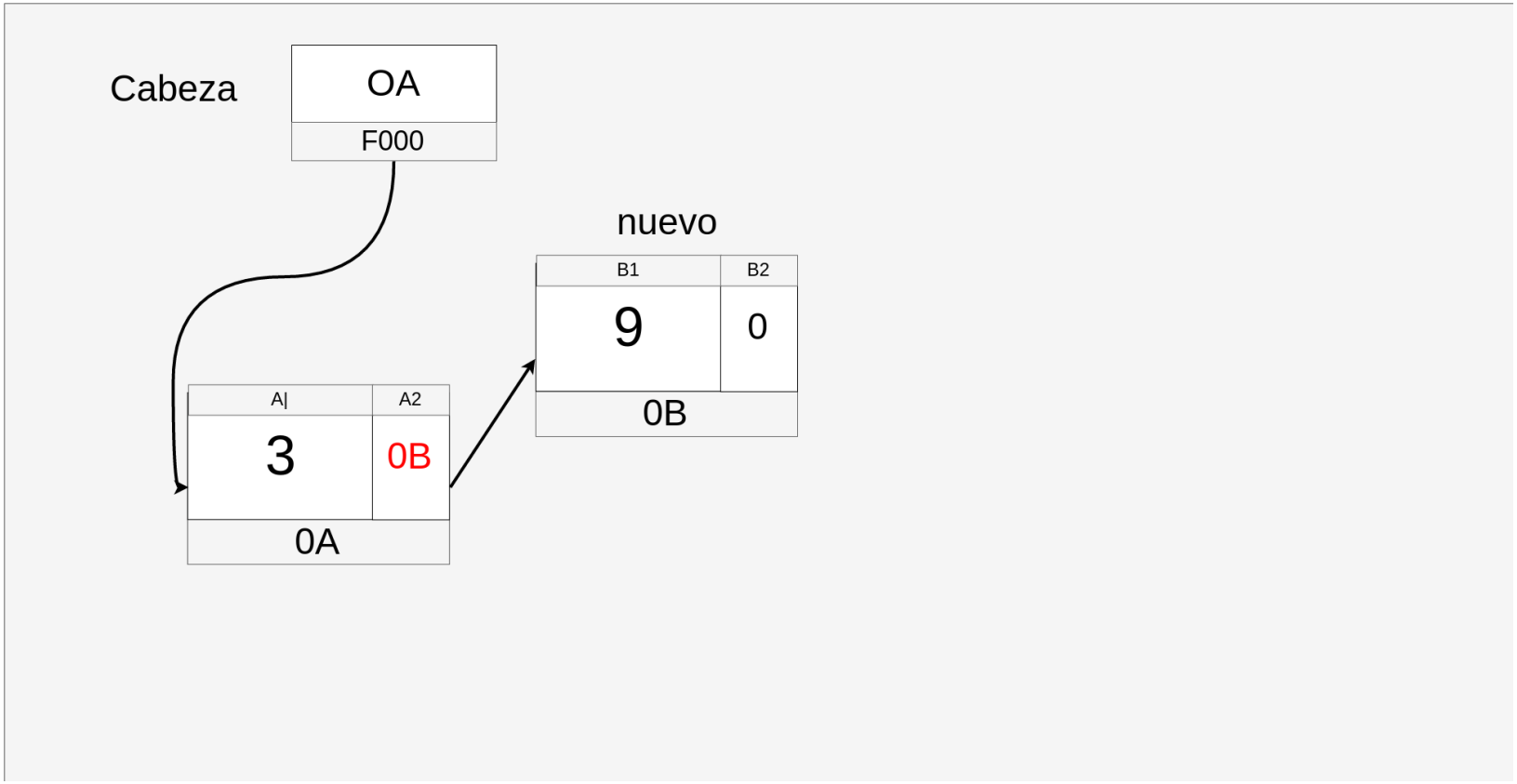


1.-Creamos un nuevo nodo con el dato a insertar (9).

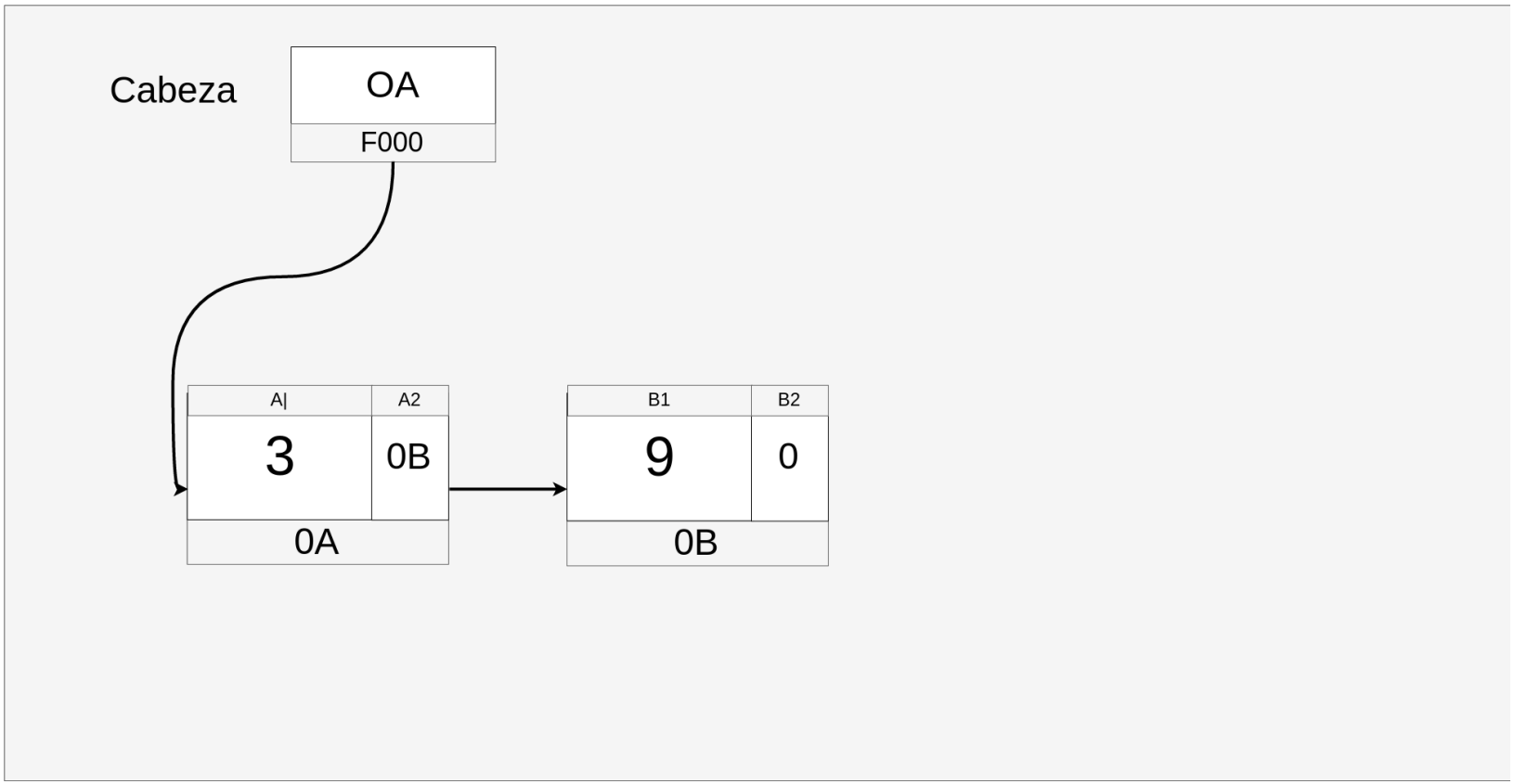
2.-Ubicamos el nodo en el lugar donde se insertara.



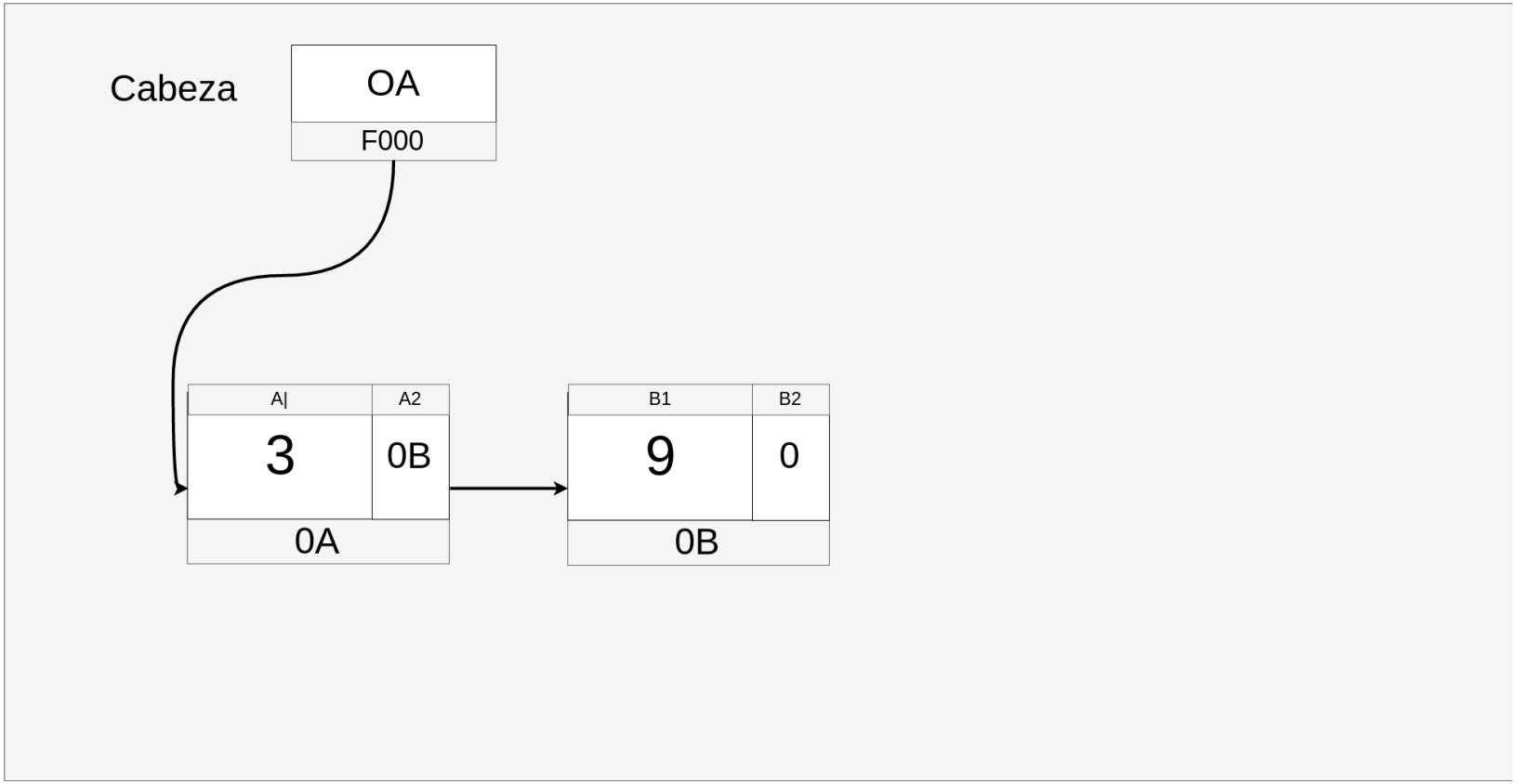
3.-Insertamos el nodo.



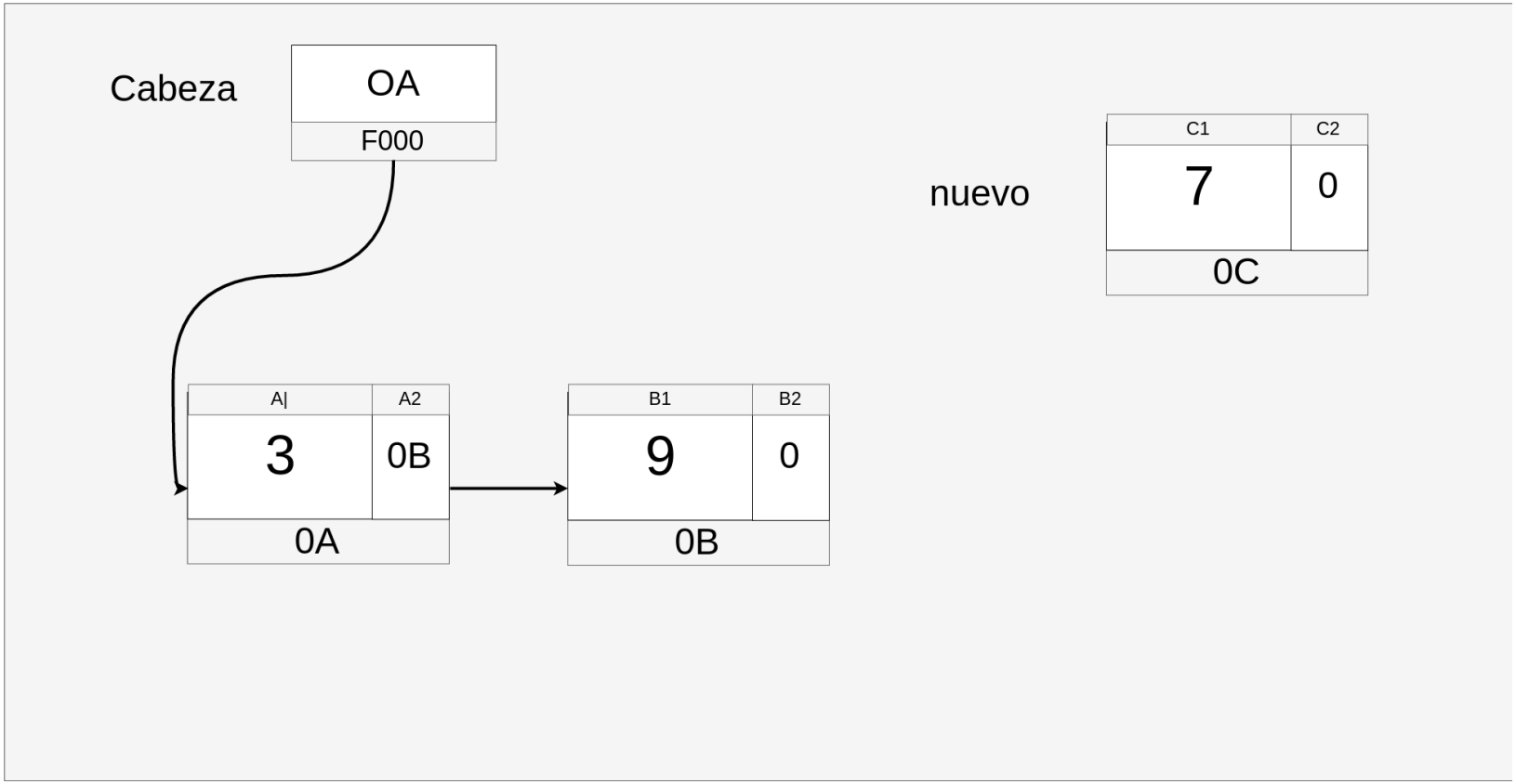
3.-Insertamos el nodo.



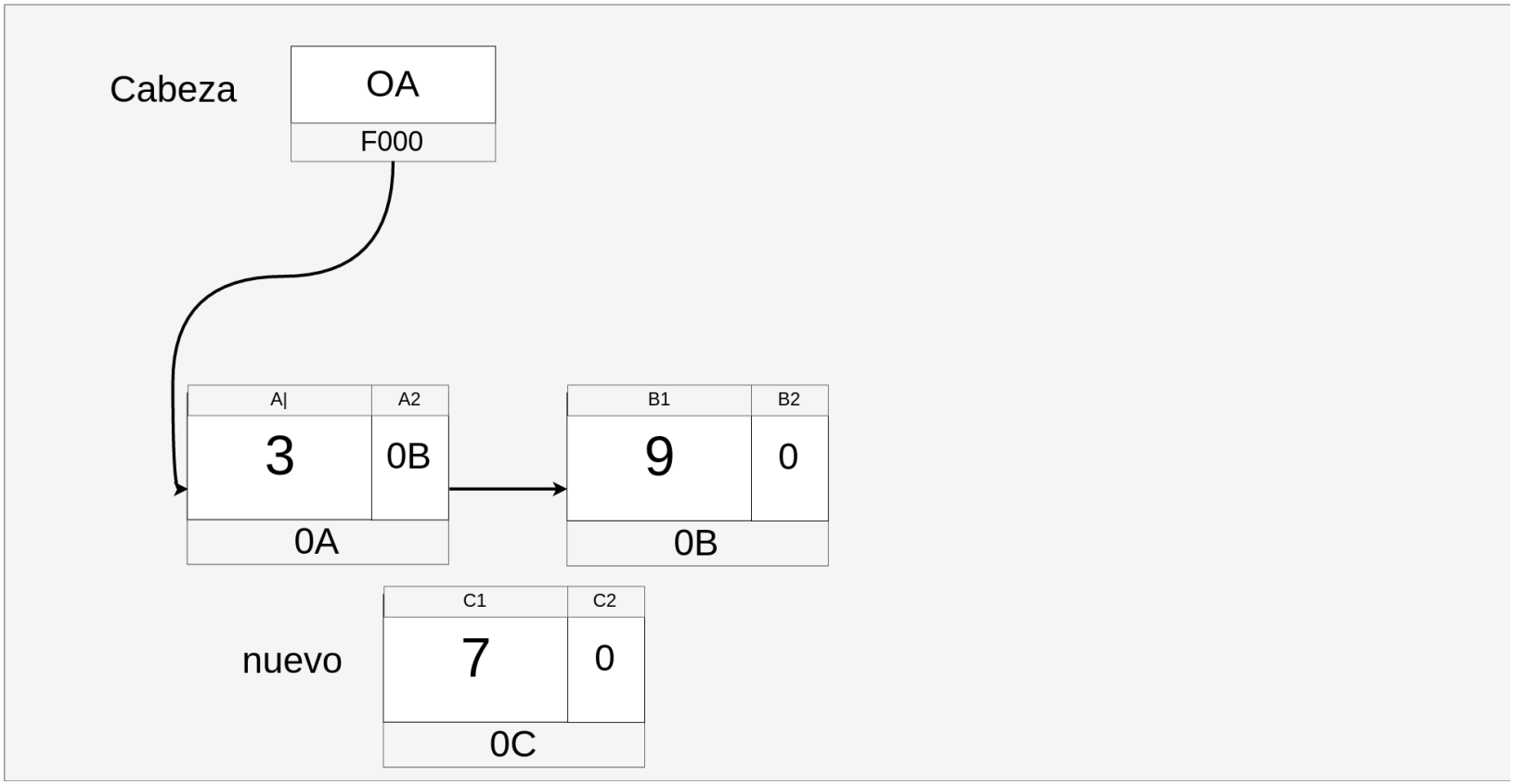
Inserción cuando el puntero a la cabeza es diferente de 0 y el dato a insertar es mayor que el dato de la cabeza



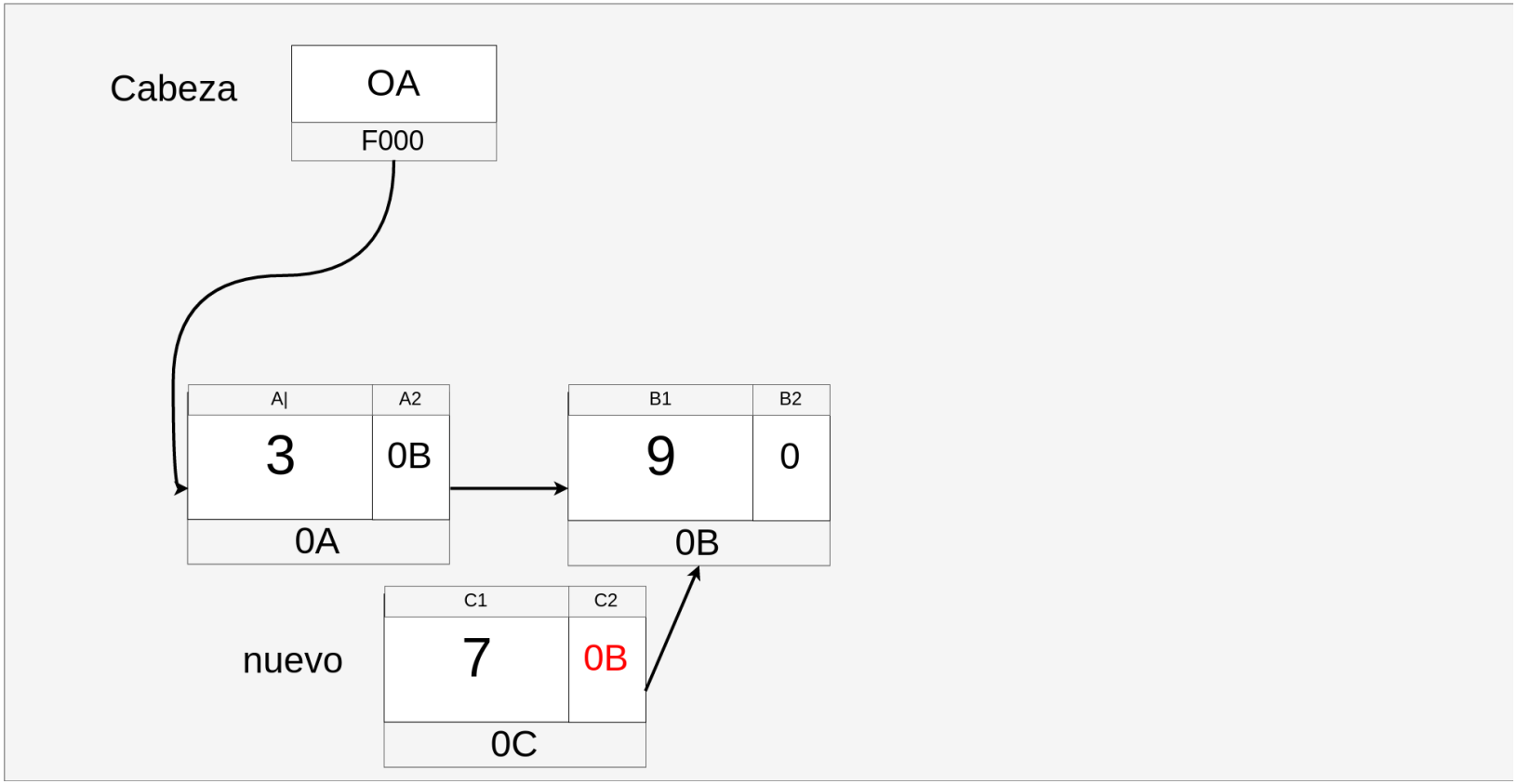
1.-Creamos un nuevo nodo con el dato a insertar (7).



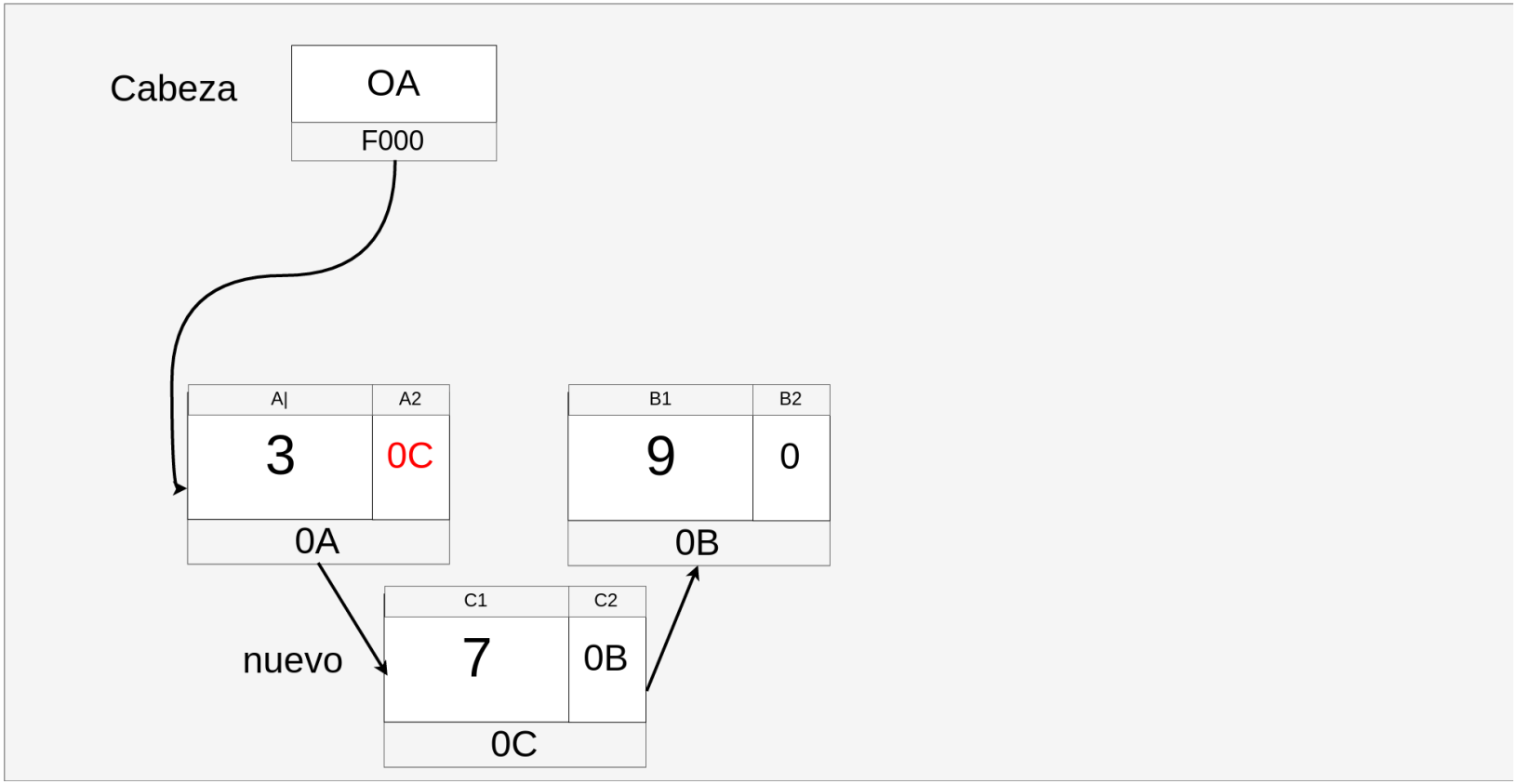
2.-Ubicamos el nodo en el lugar donde se insertara.



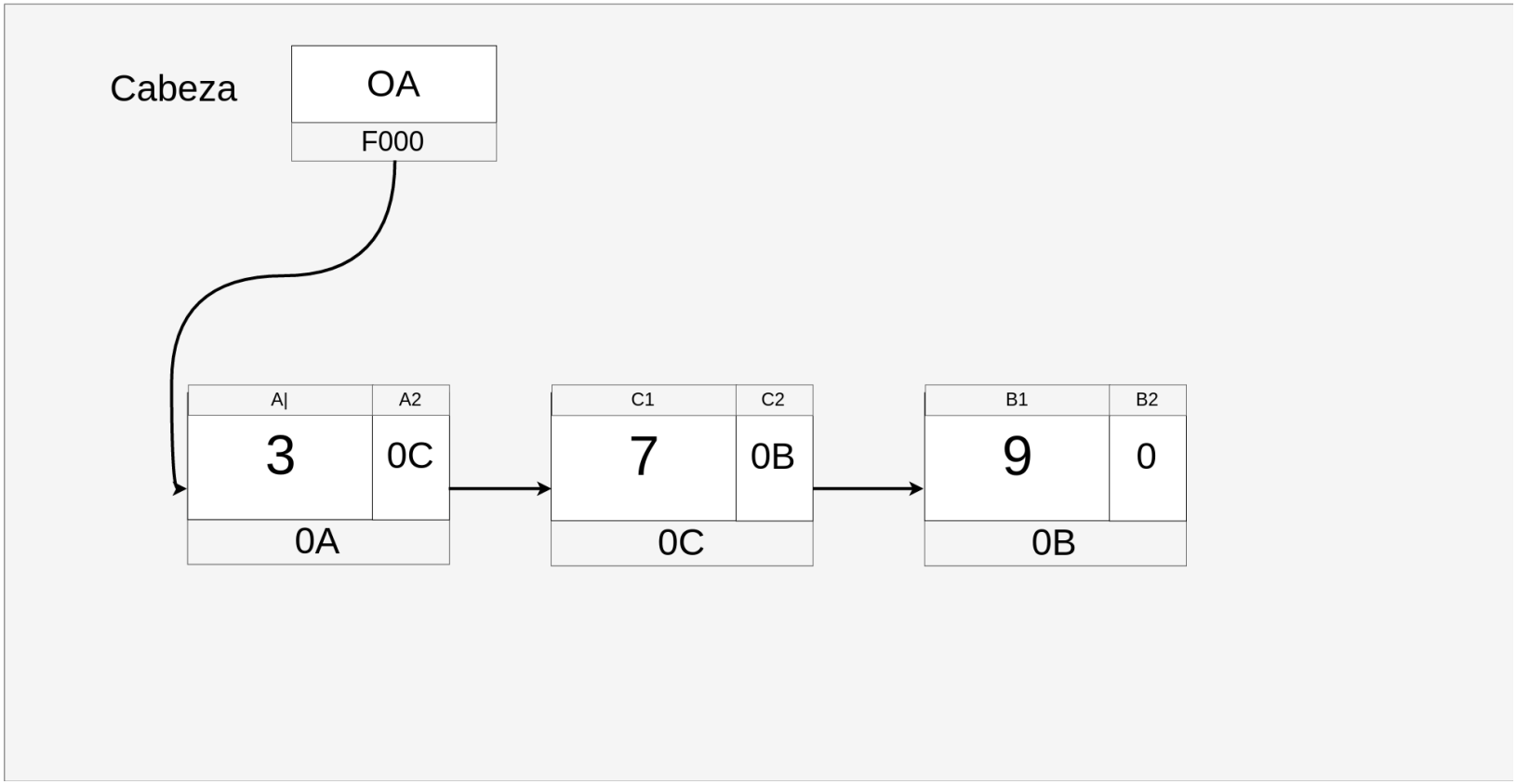
3.-Insertamos el nodo.



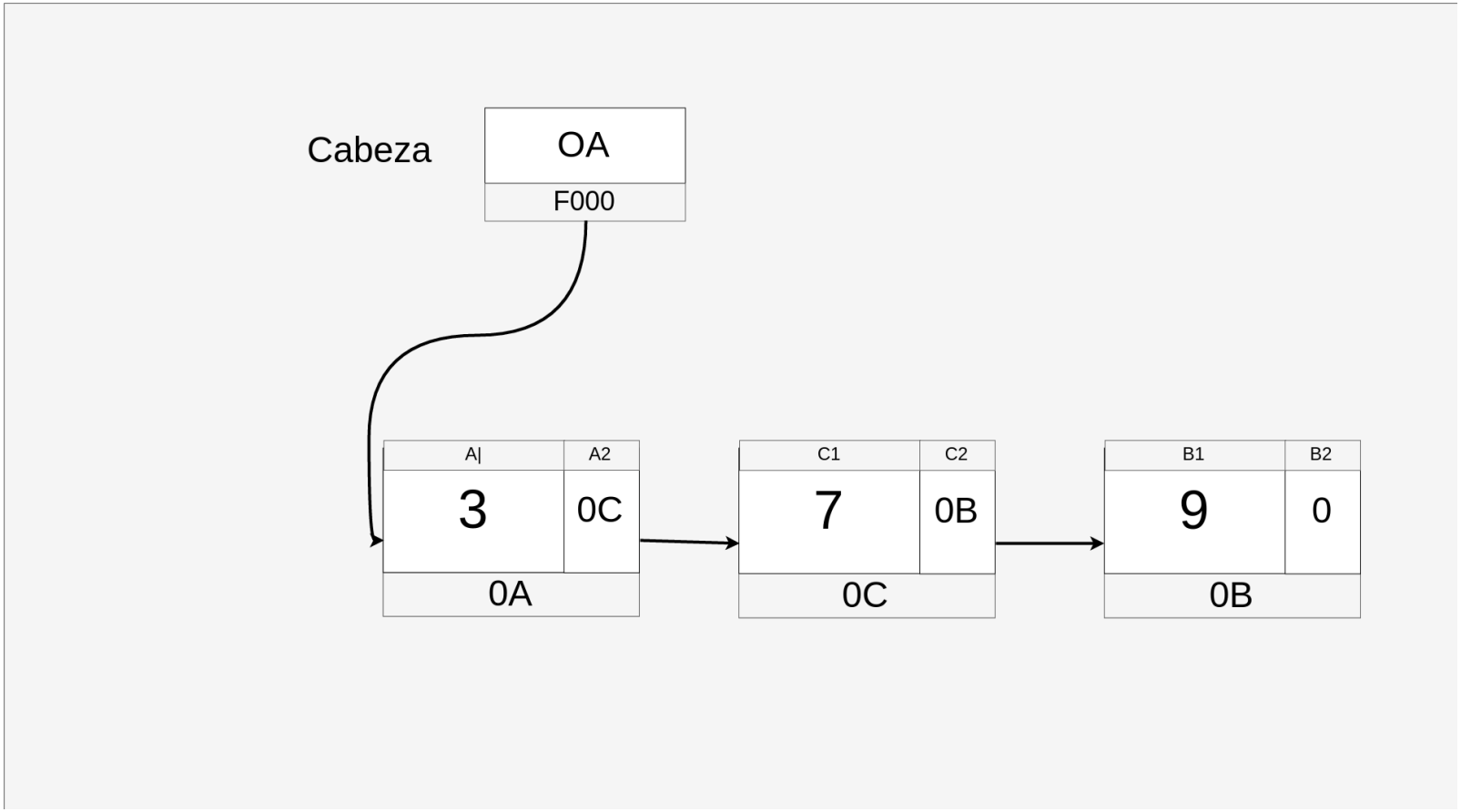
3.-Insertamos el nodo.



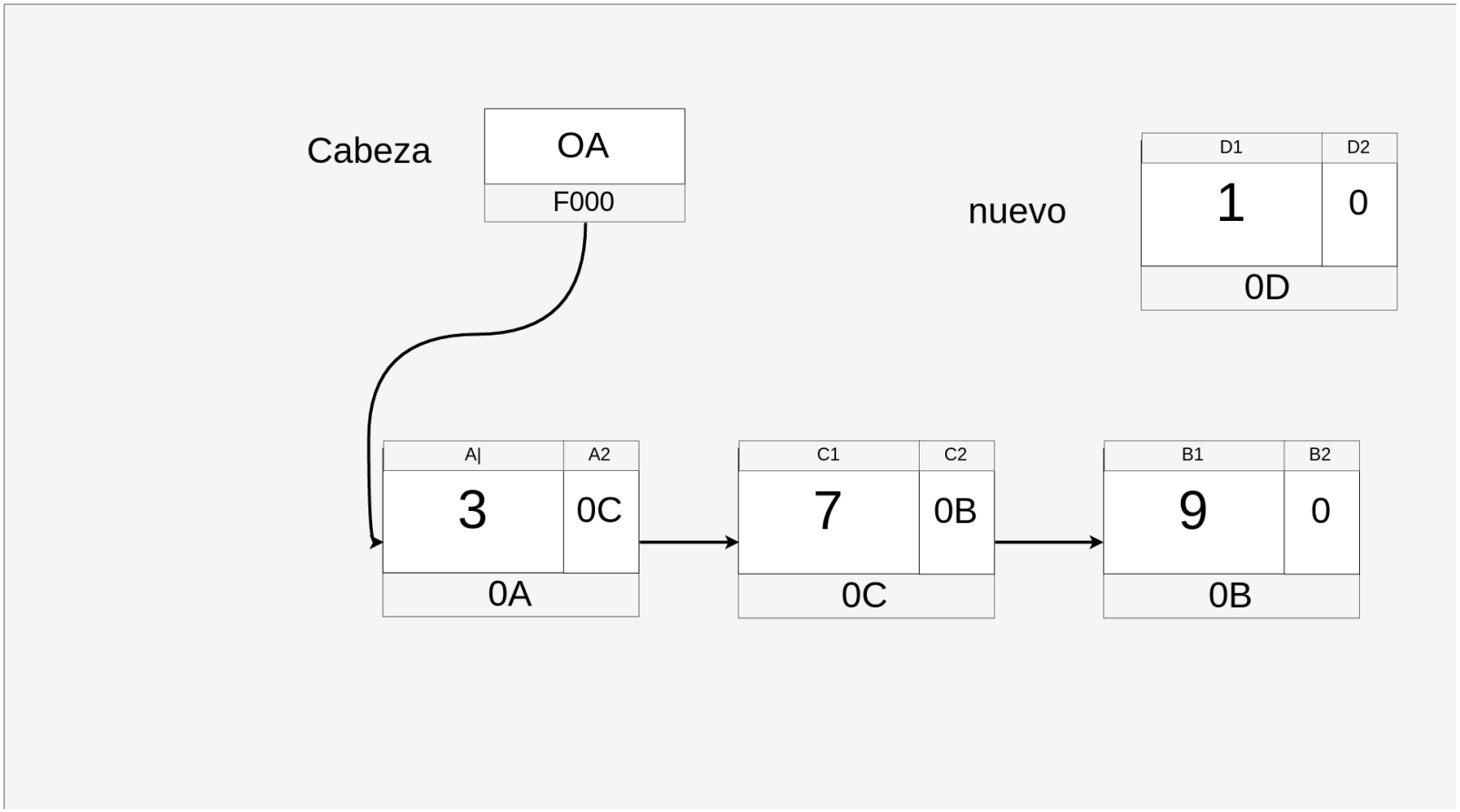
3.-Insertamos el nodo.



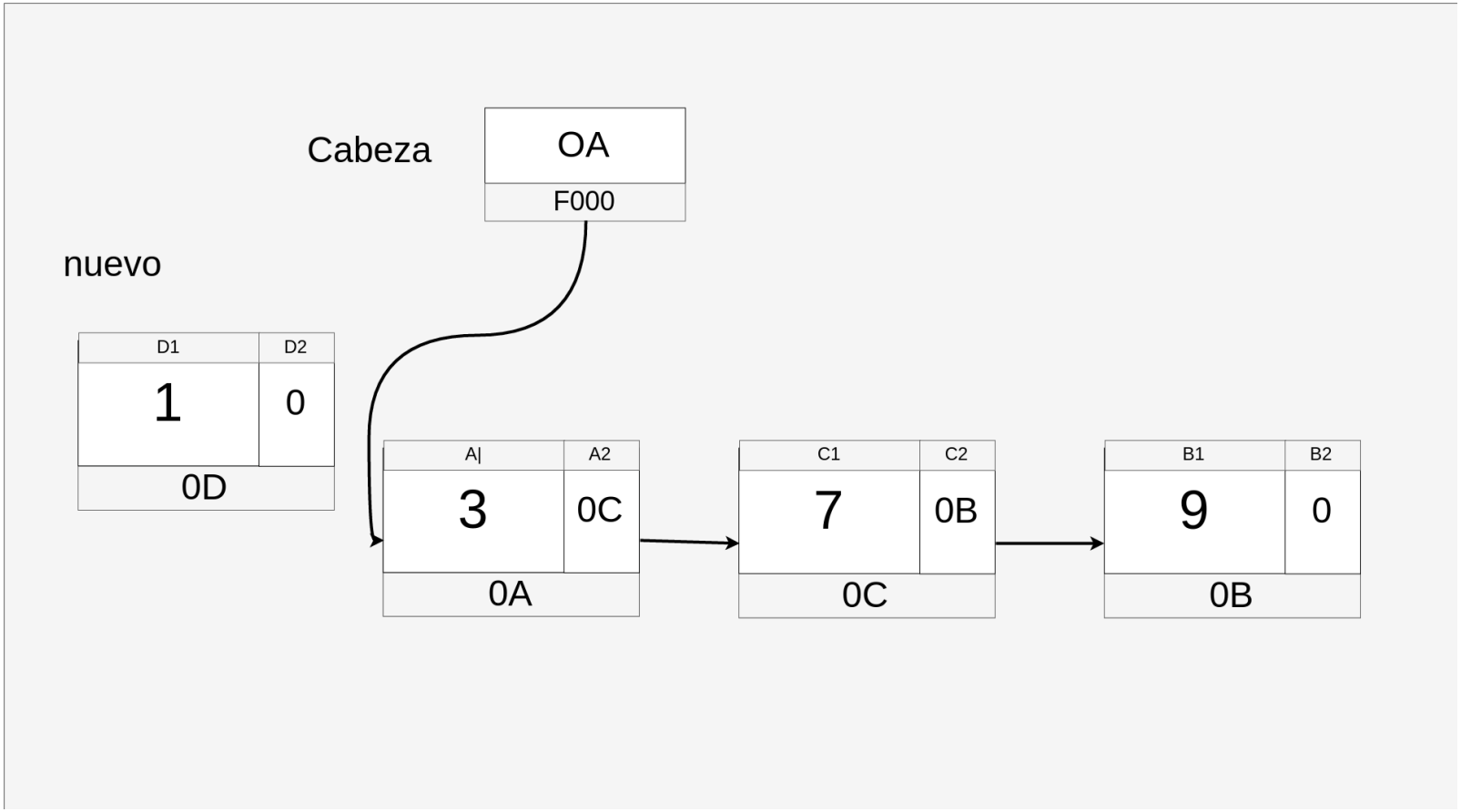
Inserción cuando el puntero a la cabeza es diferente de 0 y el dato a insertar es menor que el dato de la cabeza



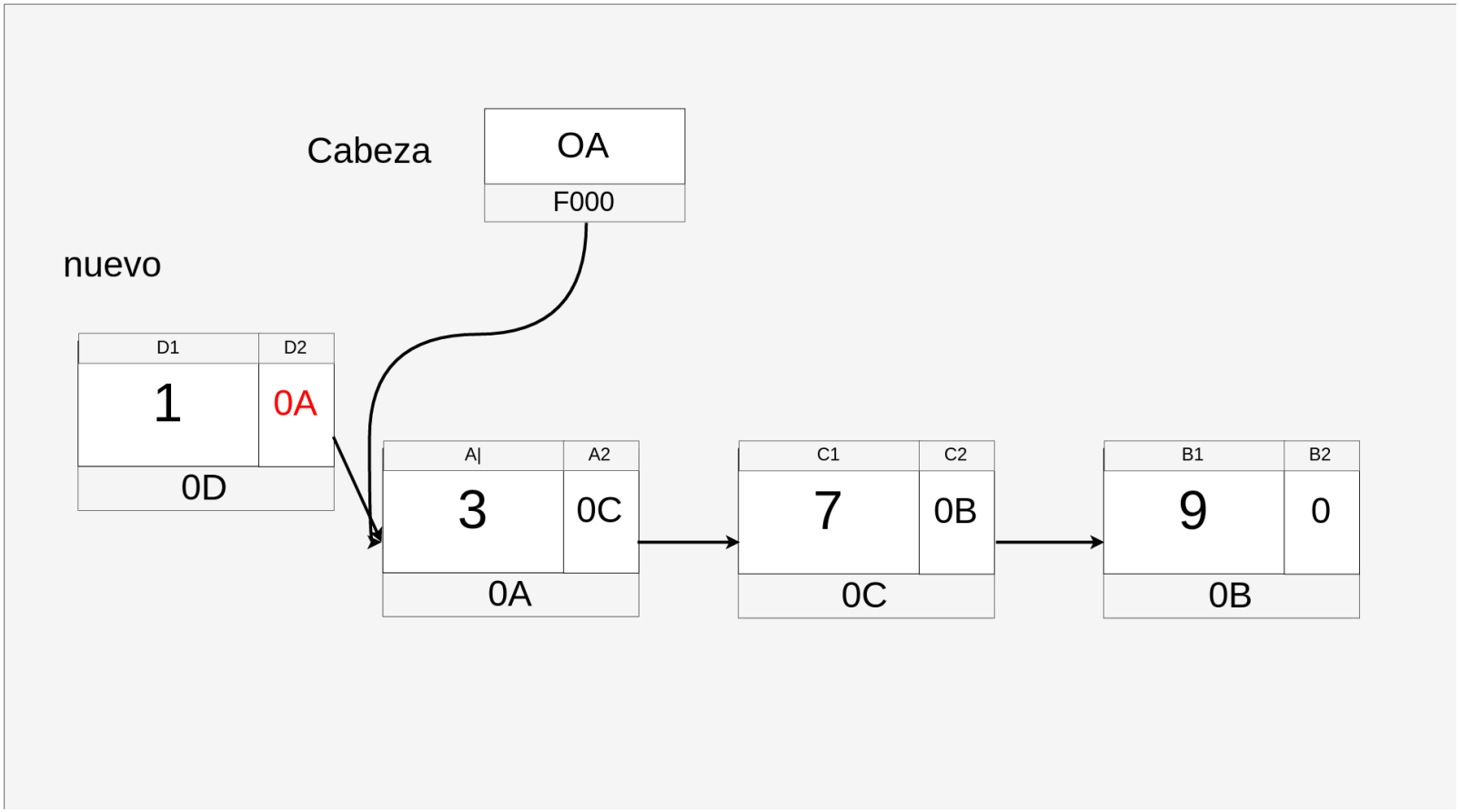
1.-Creamos un nuevo nodo con el dato a insertar (1).



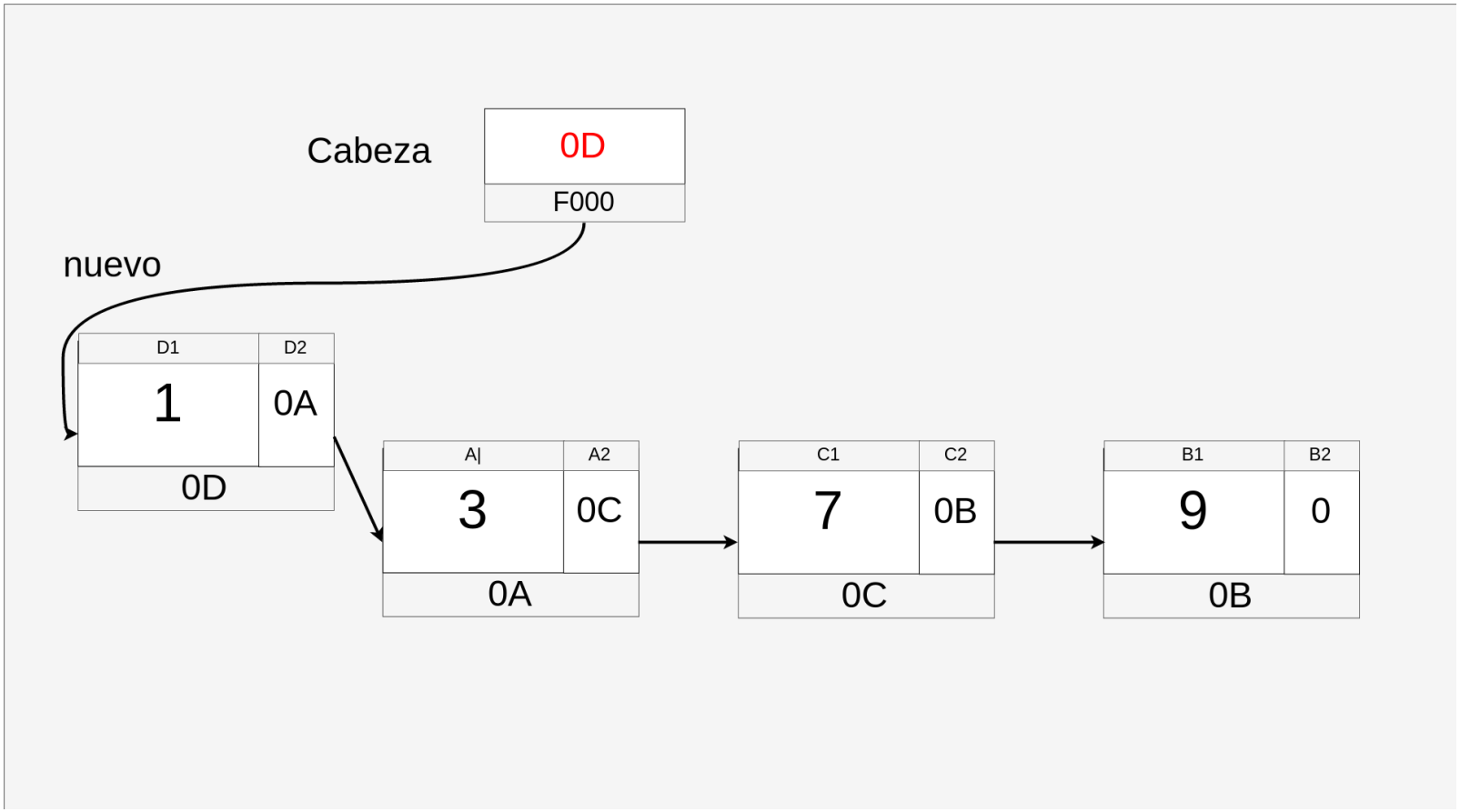
2.-Ubicamos el nodo en el lugar donde se insertara.



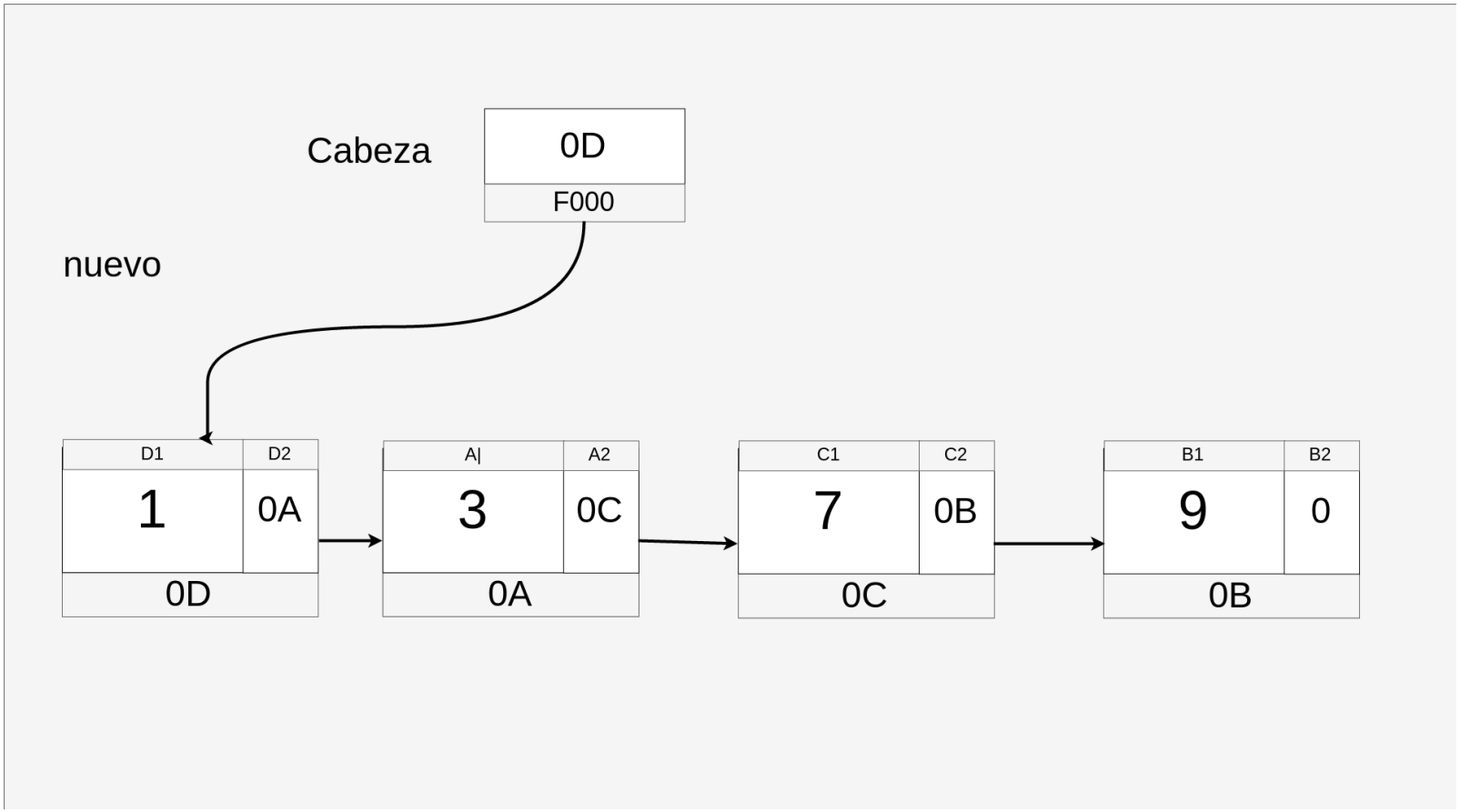
3.-Insertamos el nodo.



3.-Insertamos el nodo.



3.-Insertamos el nodo.



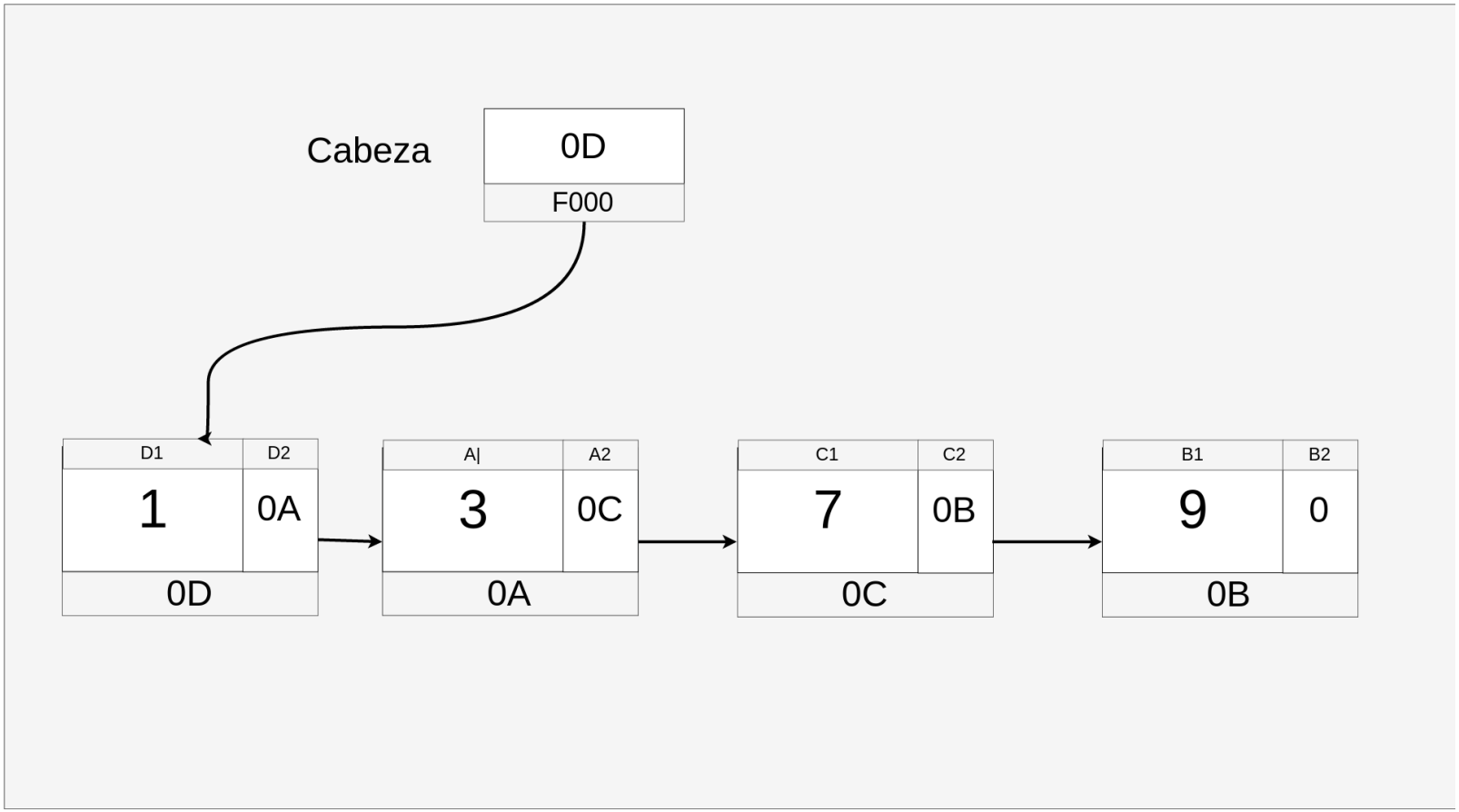
Find.

```
1   template<class T>
2   bool Lista<T>::find(T d, Nodo<T>**& p)
3   {
4       p=&m_pHead;
5       while(*p)
6       {
7           if((*p)->m_dato==d) return 1;
8           if((*p)->m_dato>d) return 0;
9           p=&((*p)->m_pSig);
10      }
11      return 0;
12  }
```

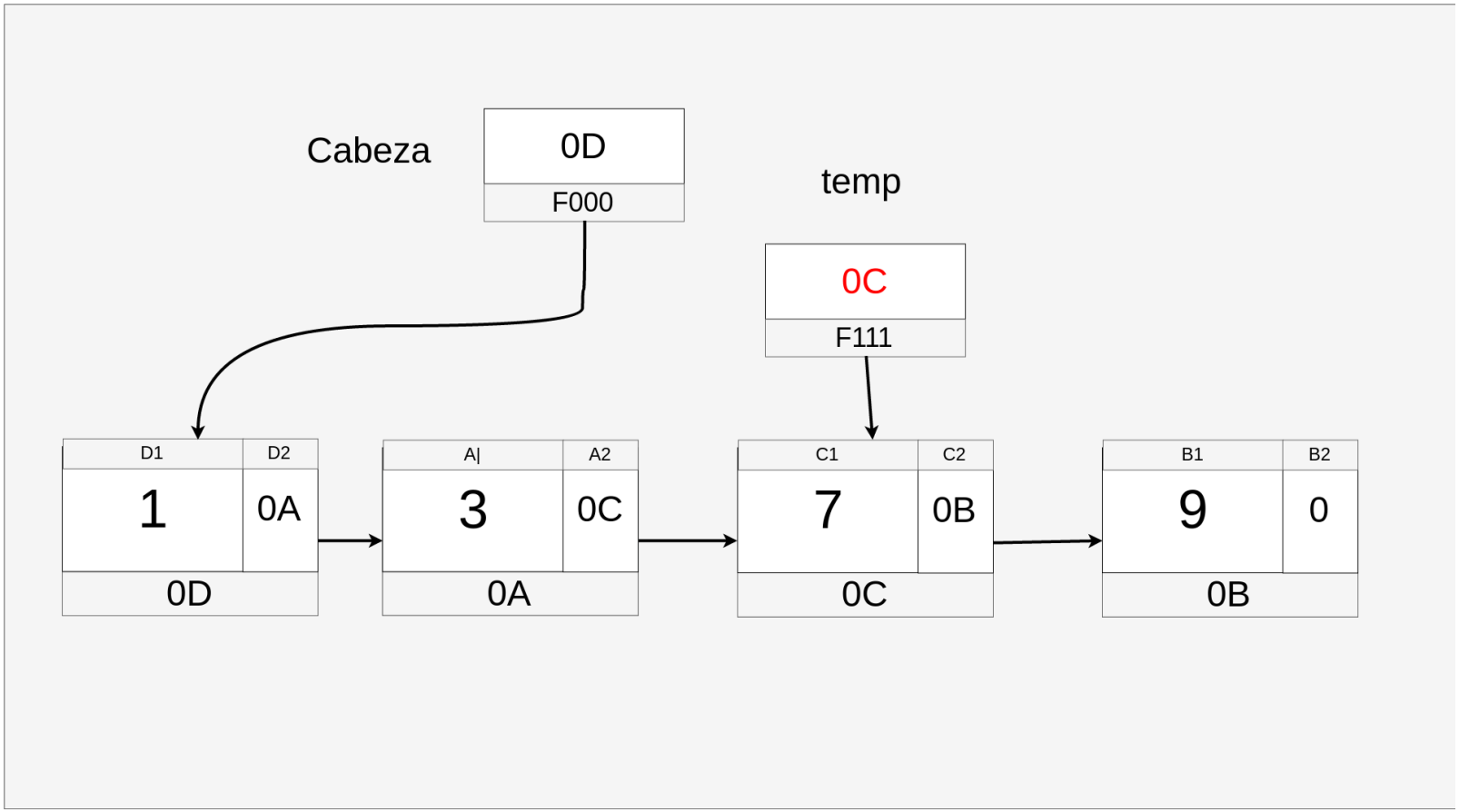
Insertar.

```
1     template<class T>
2     bool Lista<T>::add(T d)
3     {
4         Nodo<T>** p;
5         if(find(d,p)) return 0;
6         Nodo<T>* nuevo=new Nodo<T>(d);
7         nuevo->m_pSig=(*p);
8         (*p)=nuevo;
9         return 1;
10    }
```

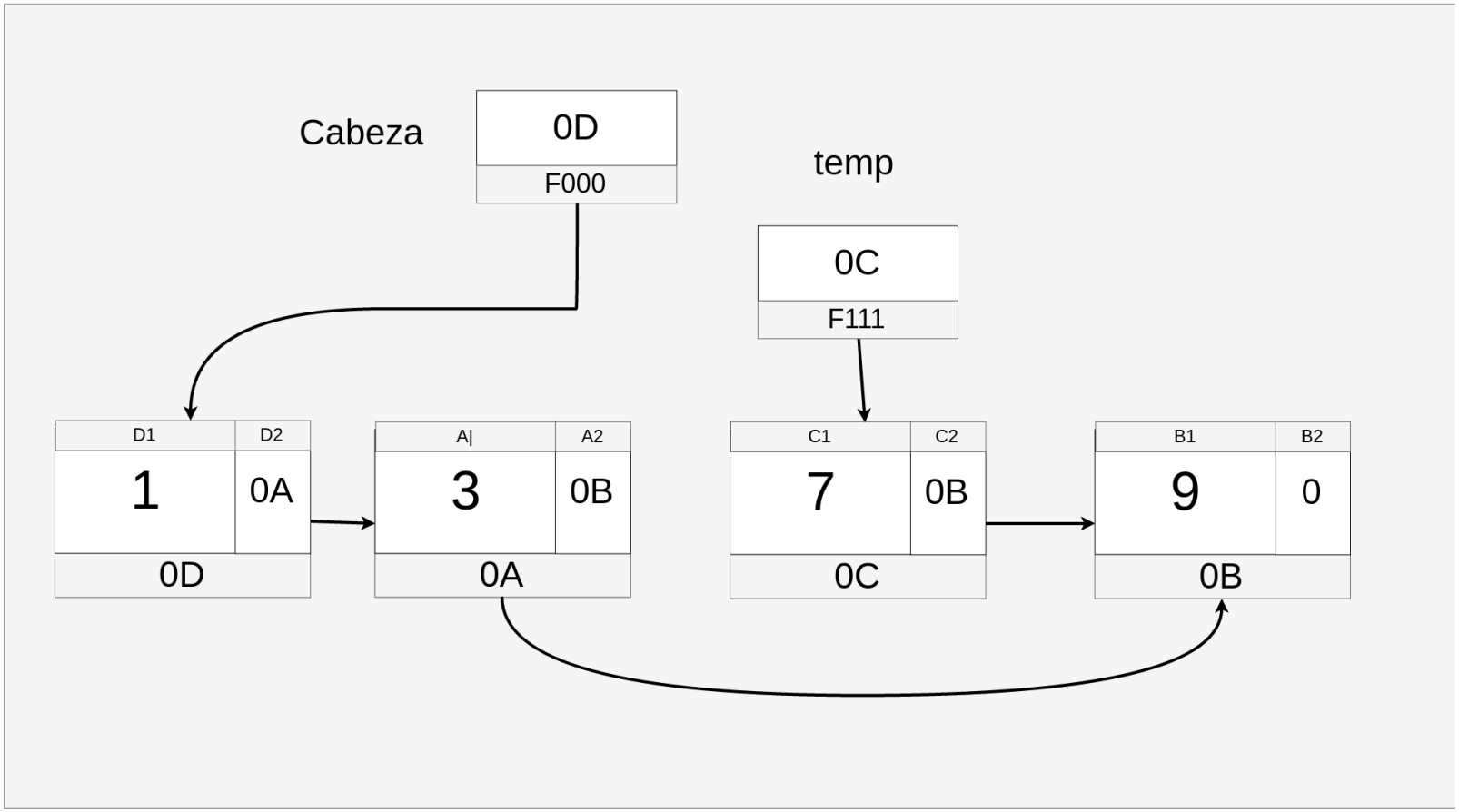
Eliminar (7).



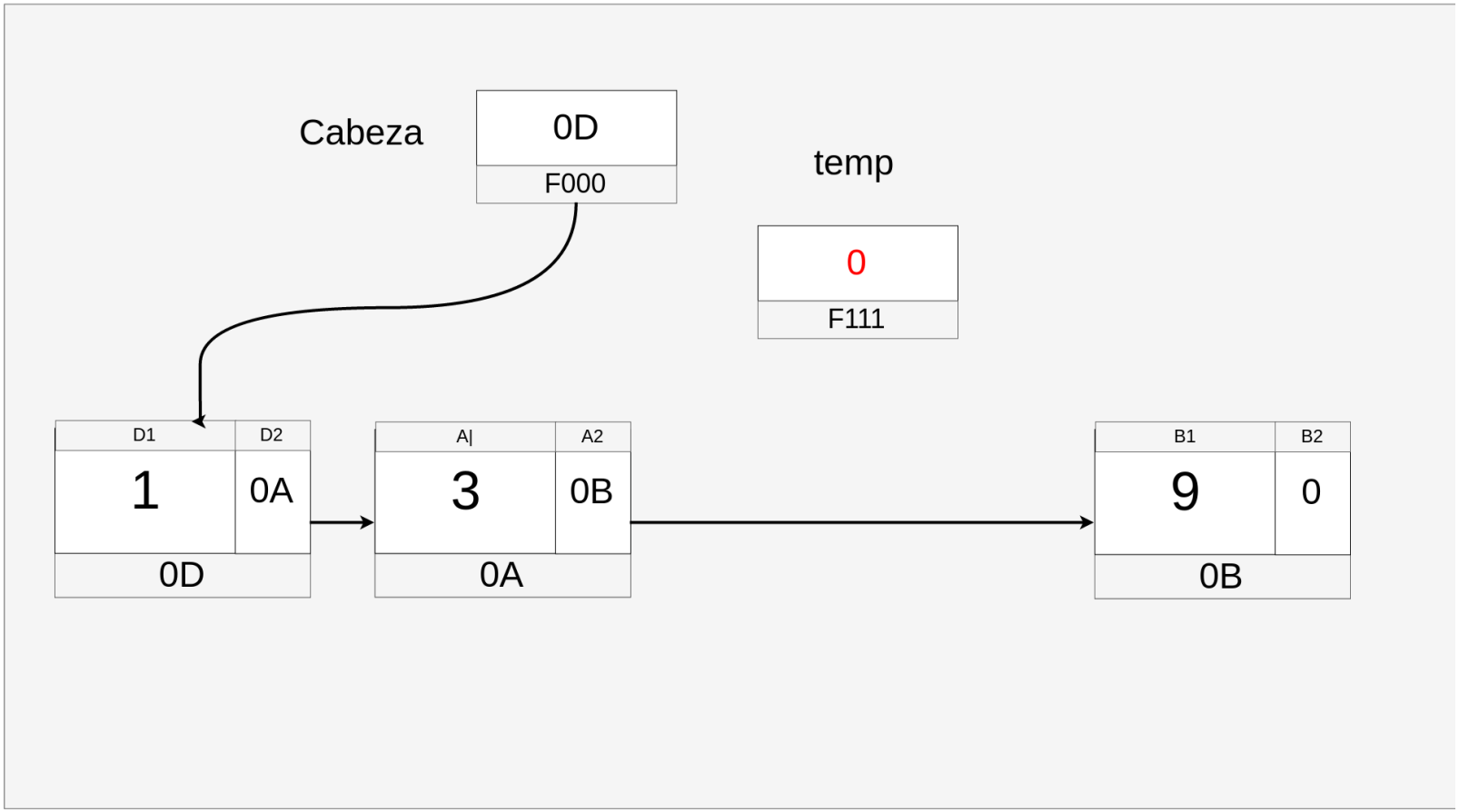
Seleccionamos el nodo que vamos a eliminar.



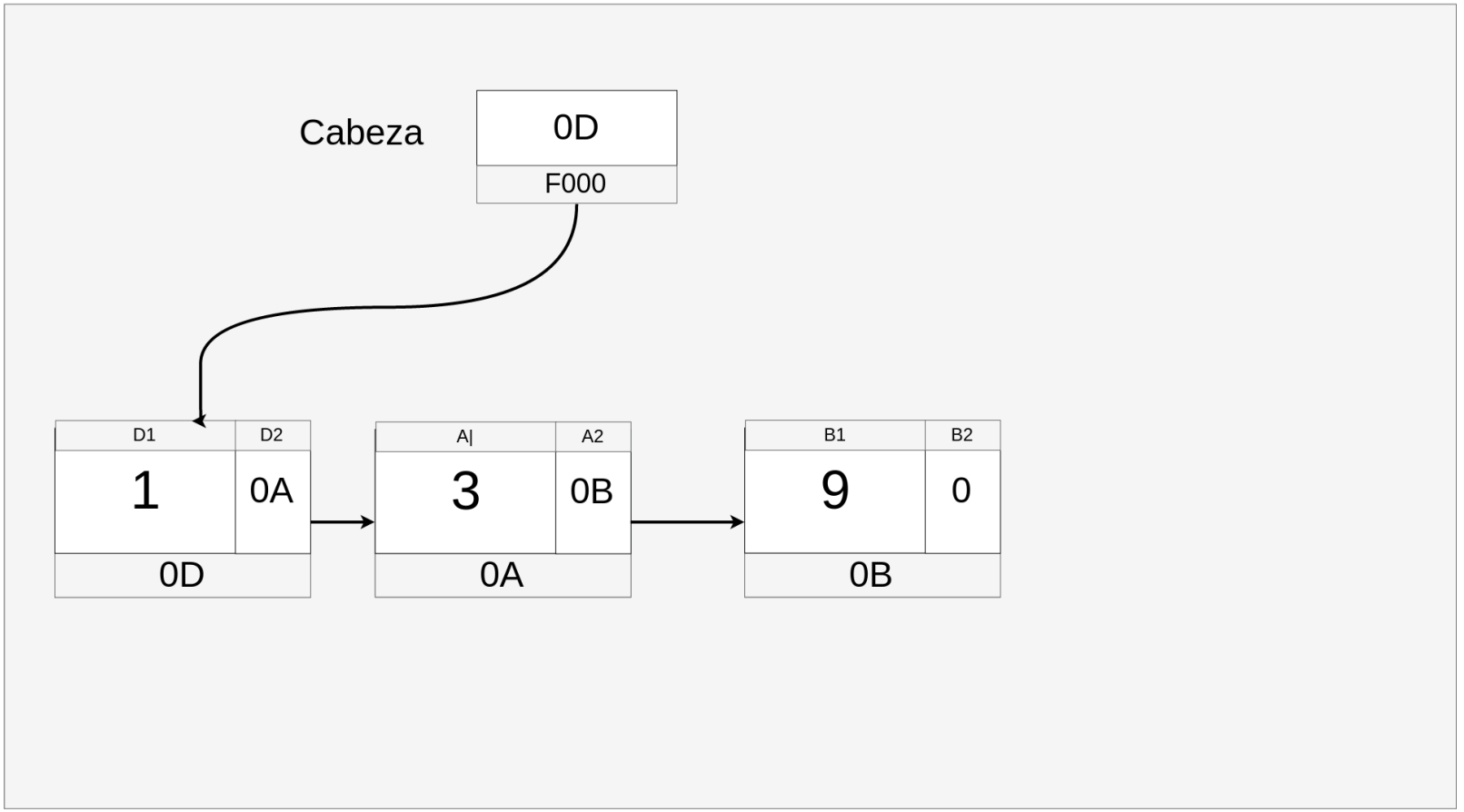
El nodo que apuntaba al nodo que eliminaremos ahora apuntara al siguiente del nodo que eliminaremos.



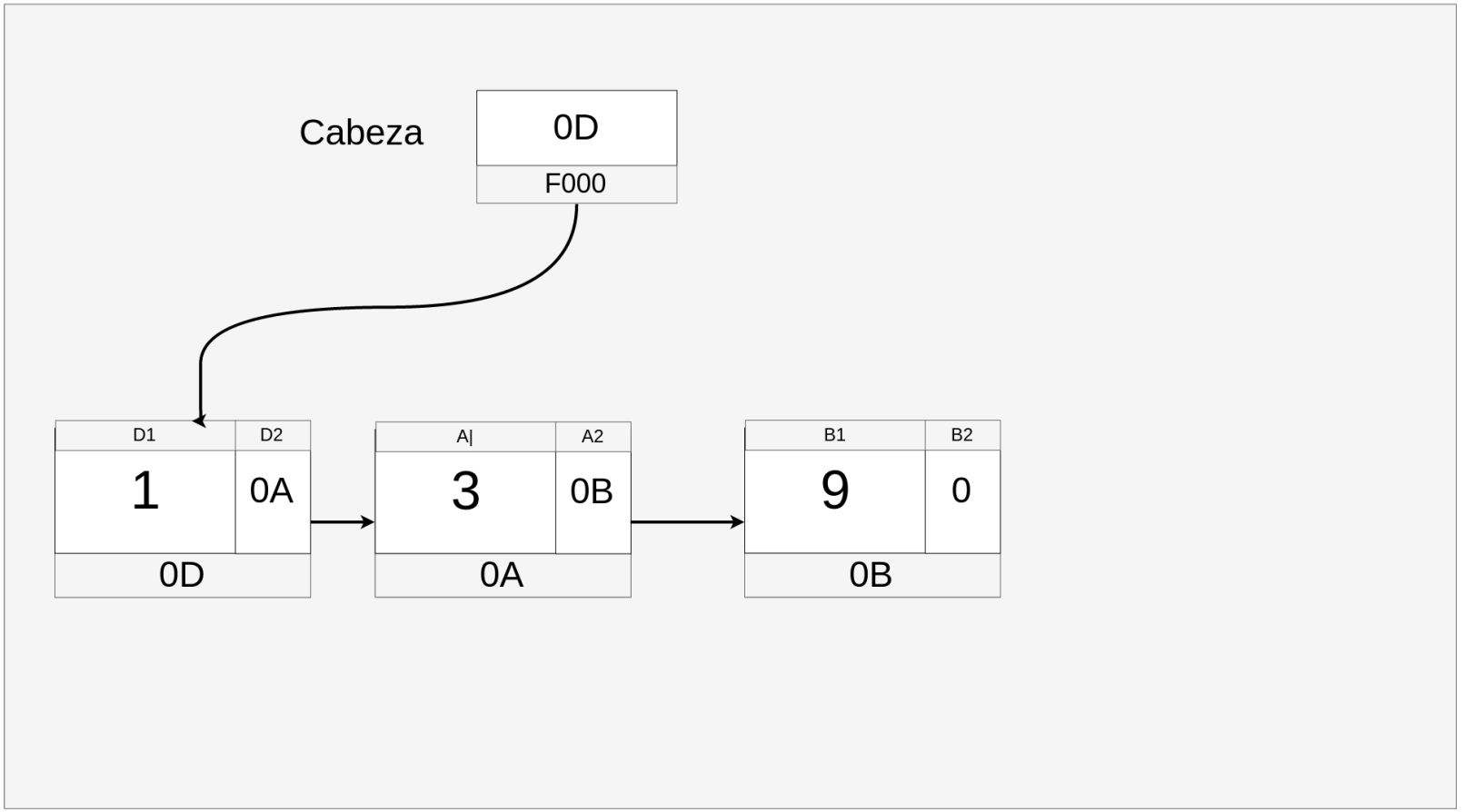
Eliminamos el nodo.



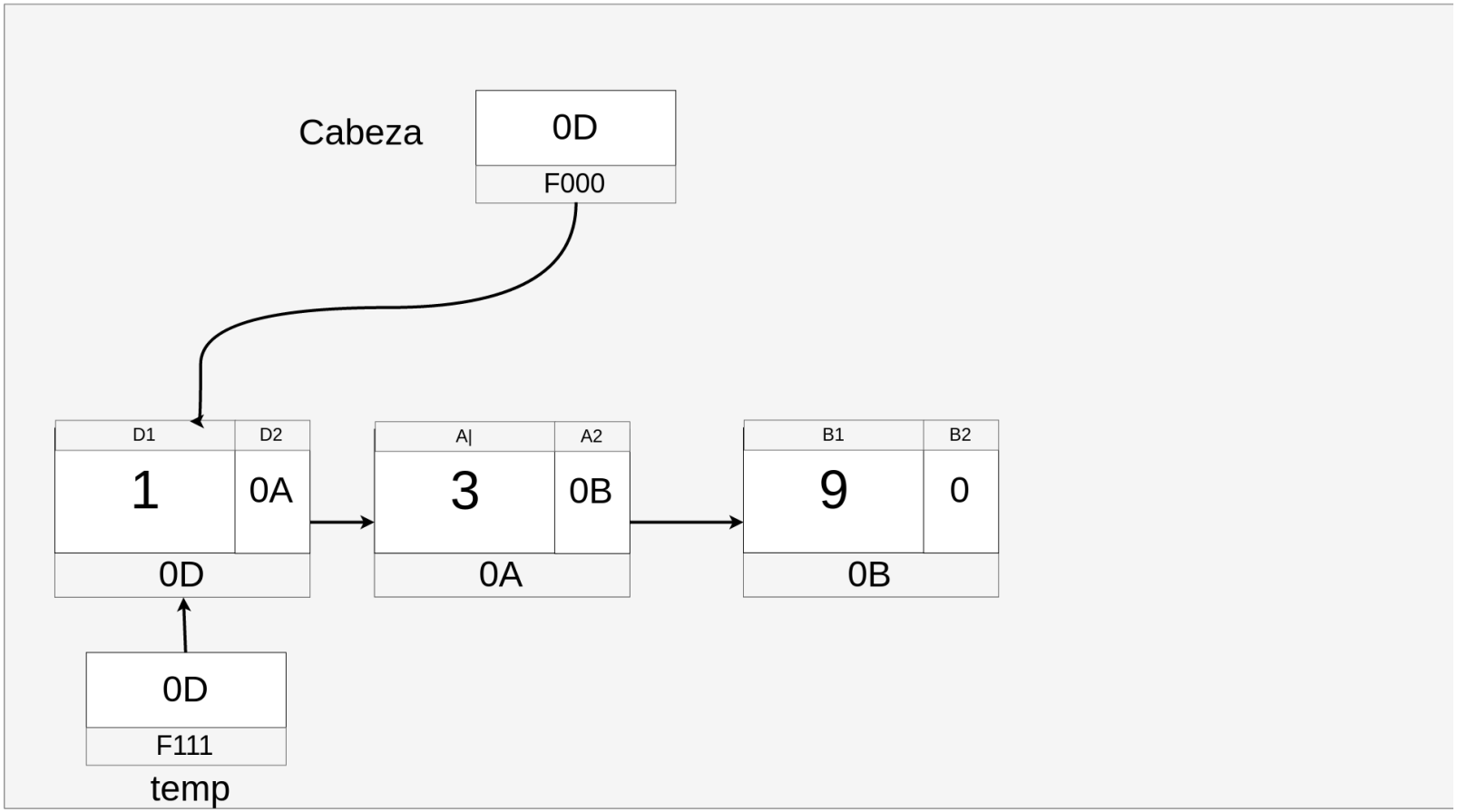
Eliminamos el nodo.



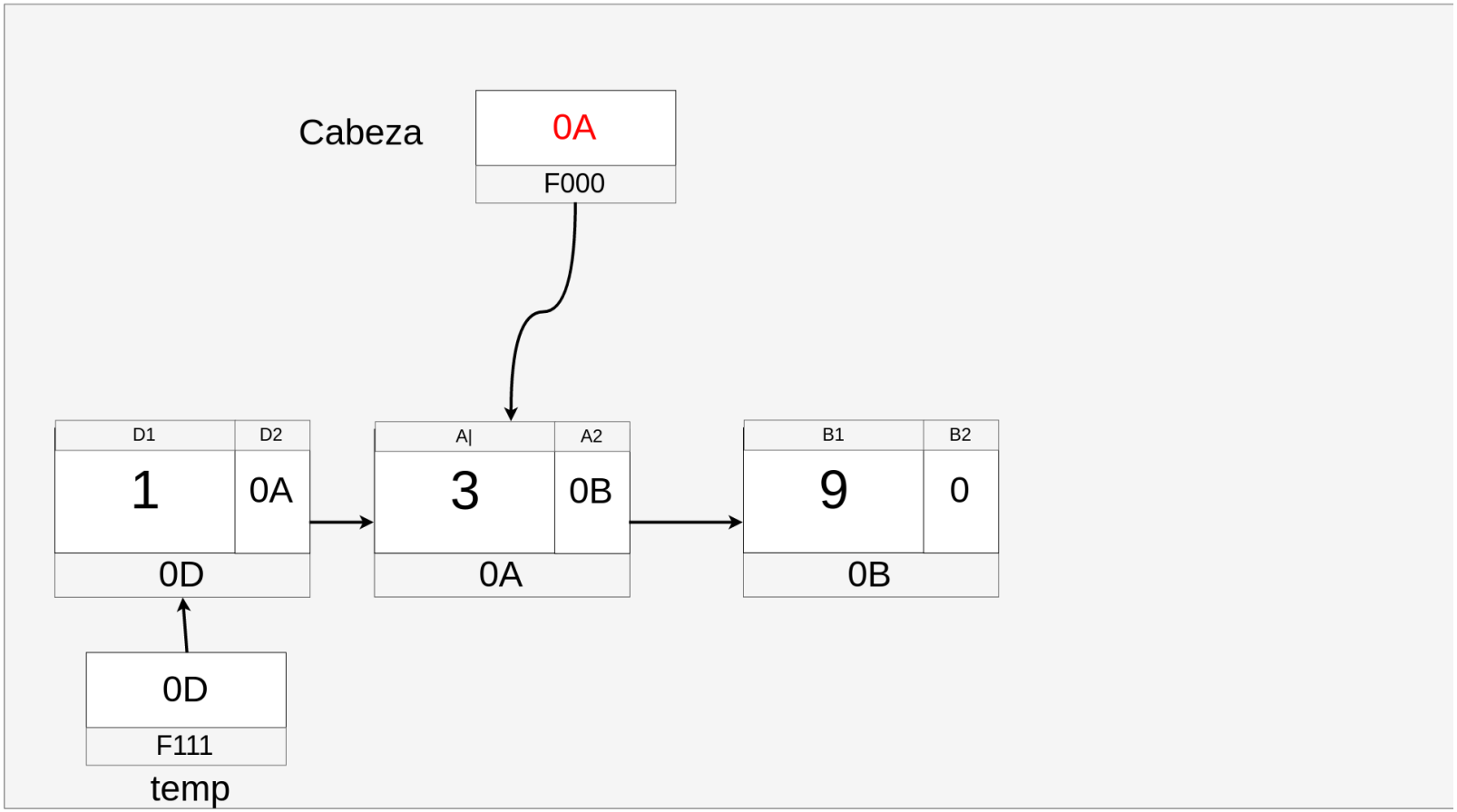
Eliminar (1).



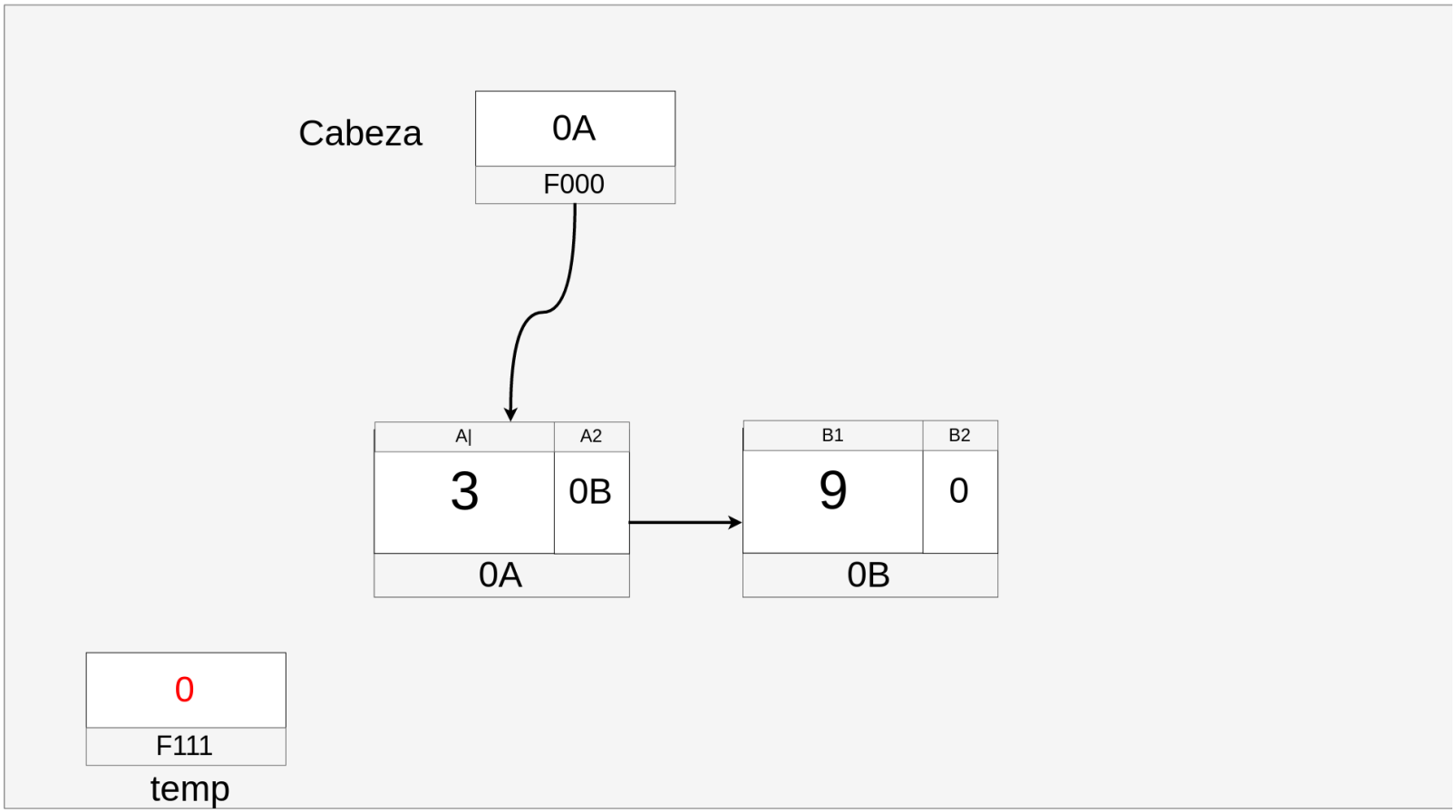
Seleccionamos el nodo que vamos a eliminar.



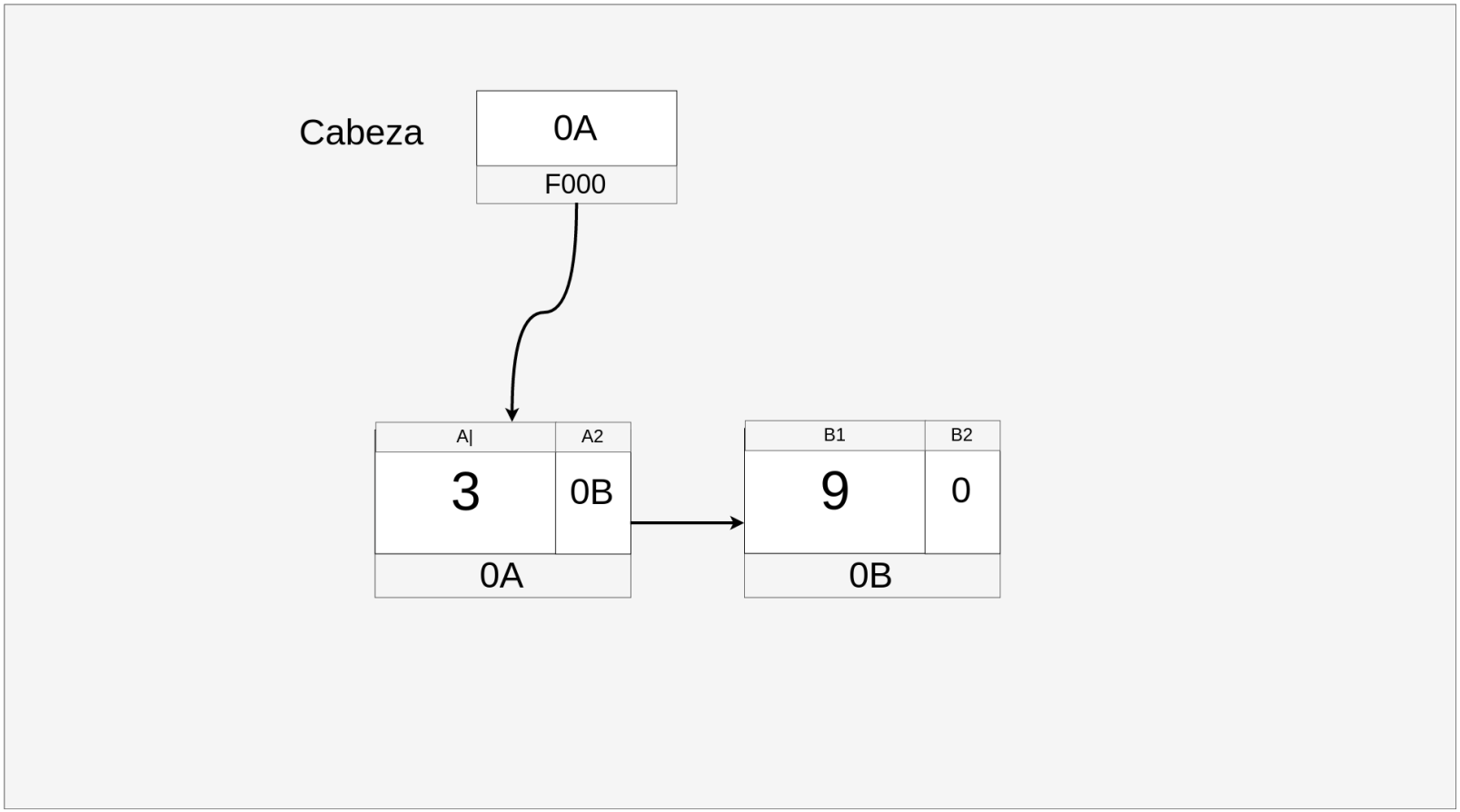
El nodo que apuntaba al nodo que eliminaremos ahora apuntara al siguiente del nodo que eliminaremos.



Eliminamos el nodo.



Eliminamos el nodo.



Eliminar.

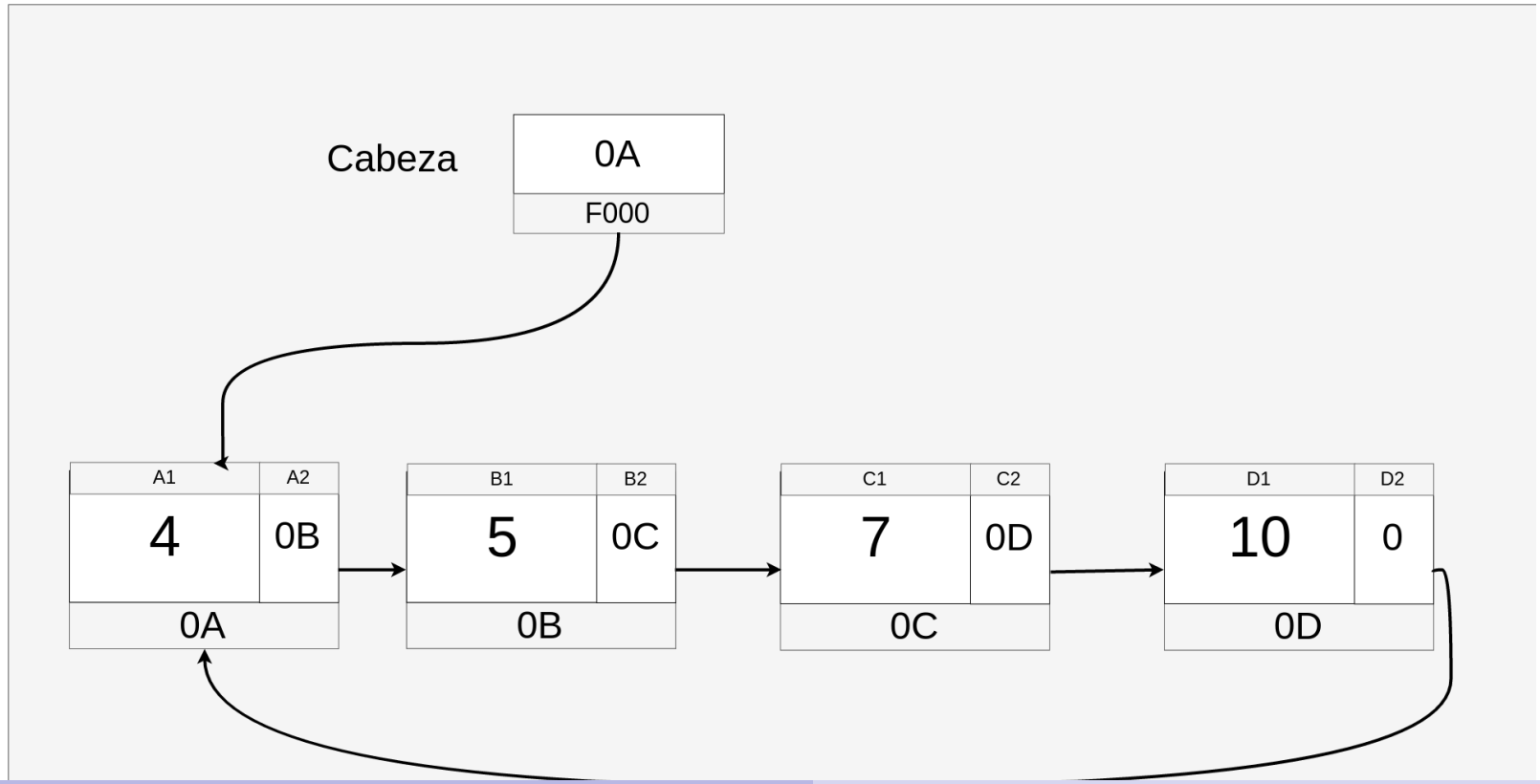
```
1     template<class T>
2     bool Lista<T>::remove(T d)
3     {
4         Nodo<T>**p;
5         if(!find(d,p)) return 0;
6         Nodo<T>* temp=(*p);
7         (*p)=temp->m_pSig;
8         delete temp;
9         return 1;
10    }
```

Imprimir.

```
1       template<class T>
2       void Lista<T>::imprimir()
3       {
4           Nodo<T>*p=m_pHead;
5           while(p){
6               cout<<p->m_dato<<endl;
7               p=p->m_pSig;
8           }
9       }
```

Lista Circular

- Una Lista Circular es una lista simple en la que el ultimo nodo apunta al primer nodo.
- Las operaciones de insertar y eliminar son similares a las de una lista simple.

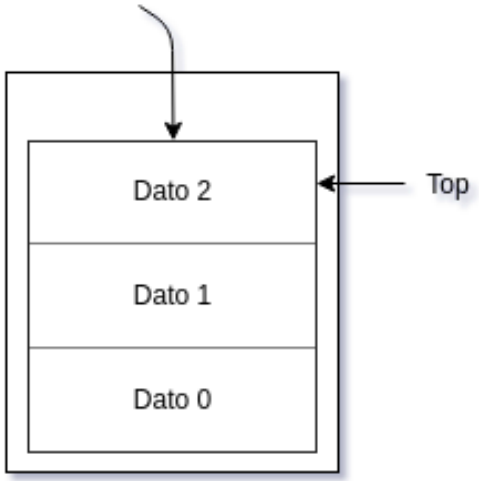


Definición

Pilas

- En una pila, el elemento borrado es el más recientemente insertado.
- La pila implementa como norma: **last-in, first-out** o **LIFO**

Figure : Esquema de una pila



Push

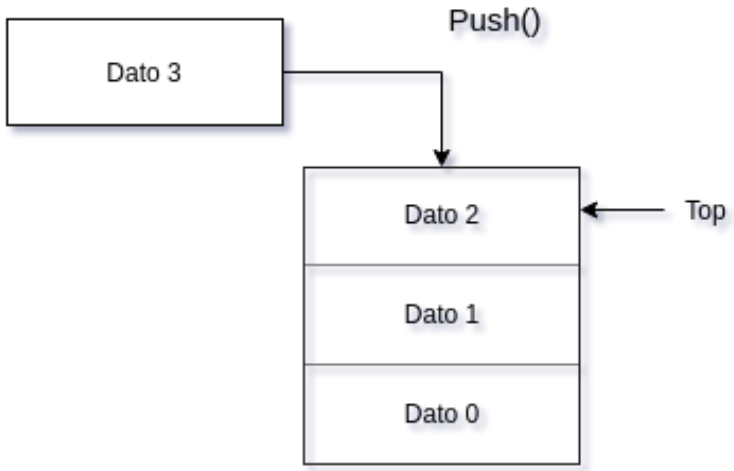


Figure : Inserción de un elemento

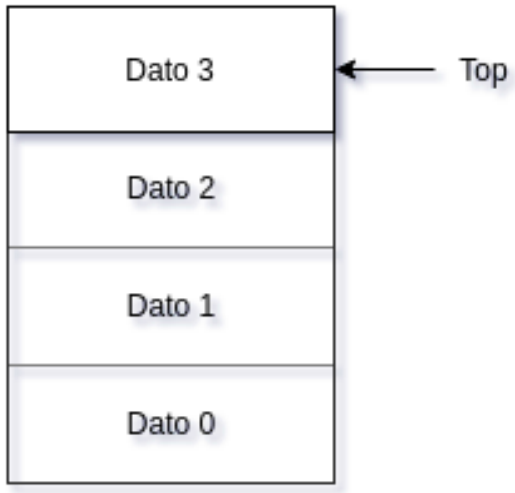


Figure : Pila despues de la inserción

Pop

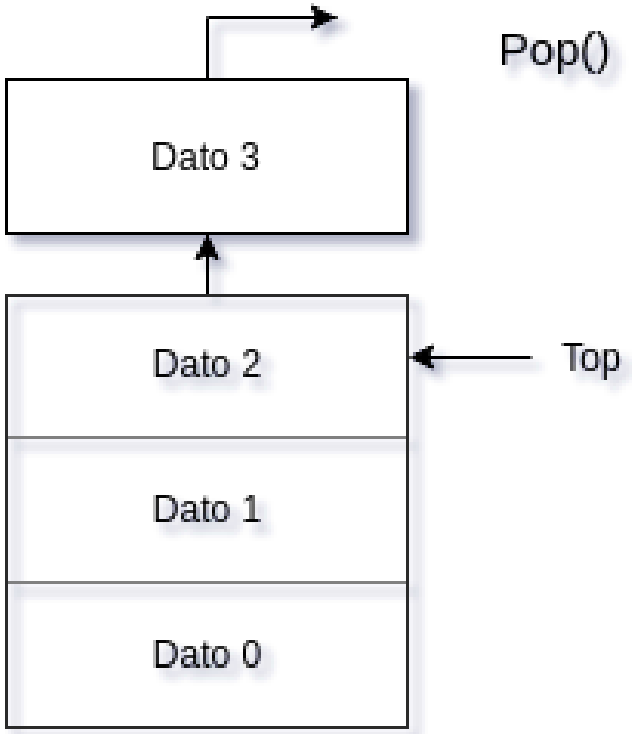


Figure : Eliminación de un elemento

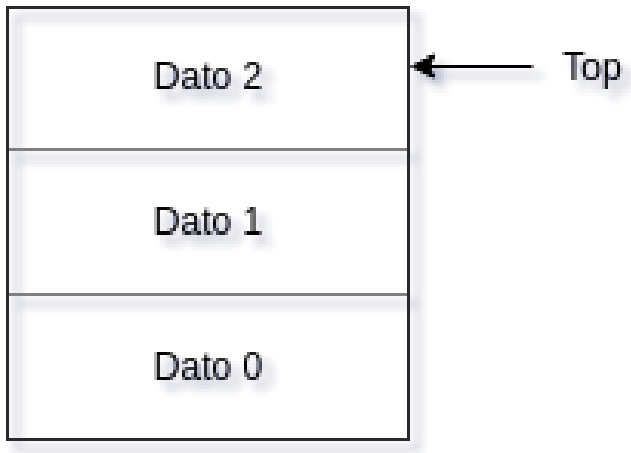


Figure : Pila despues de eliminar un elemento

Implementacion Clase Nodo

```
1  template <class T>
2  class Nodo
3  {
4  public:
5    T            m_dato;
6    Nodo<T> *    m_psiguiente;
7  public:
8    Nodo(T d){
9      m_psiguiente = 0;
10     m_dato = d;
11   }
12 };
```

Implementacion Clase Pila

```
1  #include "Nodo.h"
2  #include <ostream>
3  using namespace std;
4  template <class T>
5  class Pila
6  {public:
7    Nodo<T> *   m_phead;
8  public:
9    Pila(){
10     m_phead = 0;}
11   void push(T);
12   T    pop();
13   T    top();
14   void print(ostream & os); };
```

Push - Lista

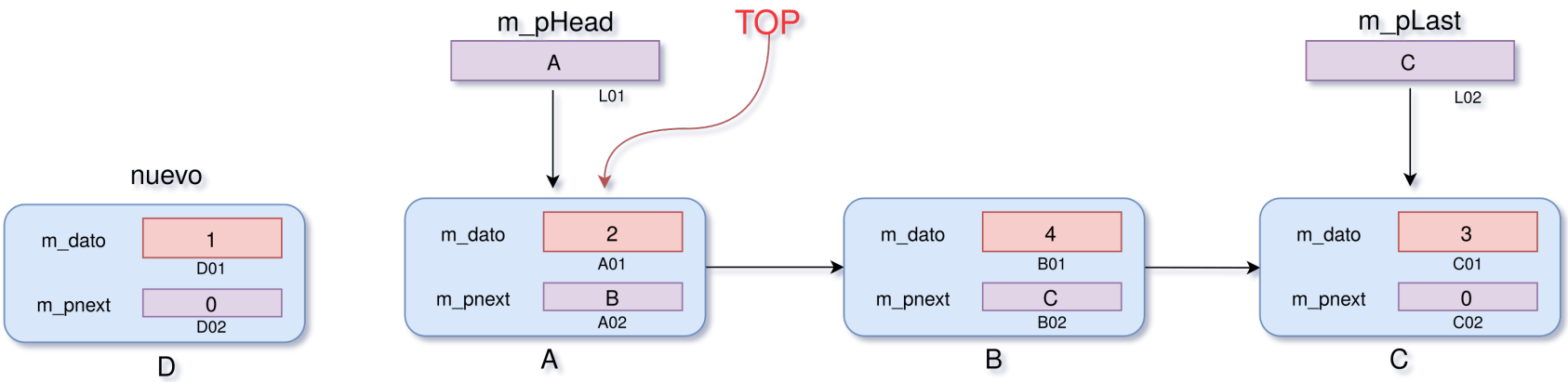


Figure : Crear nuevo Nodo

Push - Lista

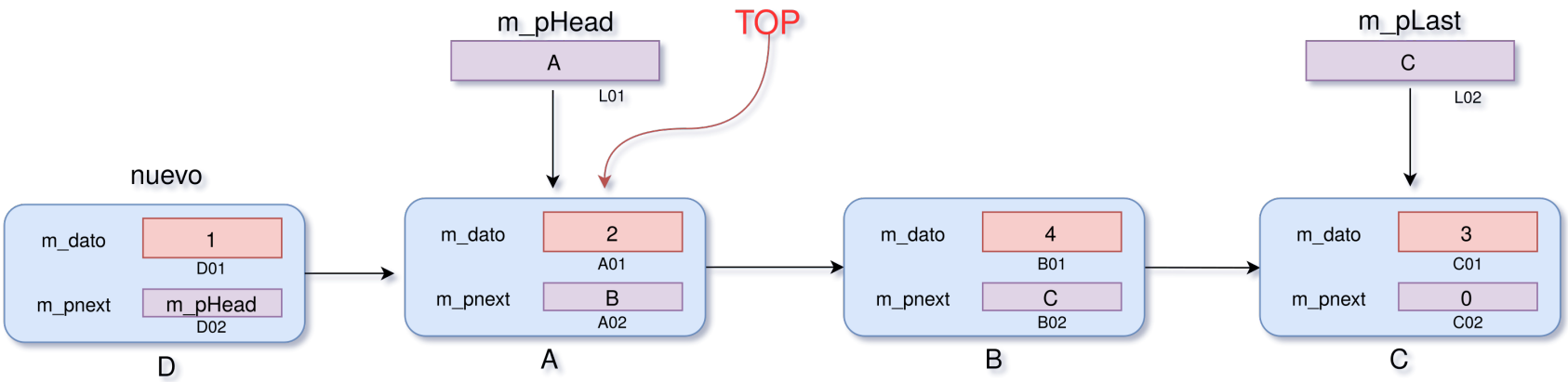


Figure : Apuntar el Siguiente del Nuevo a la Cabeza de la Lista

Push - Lista

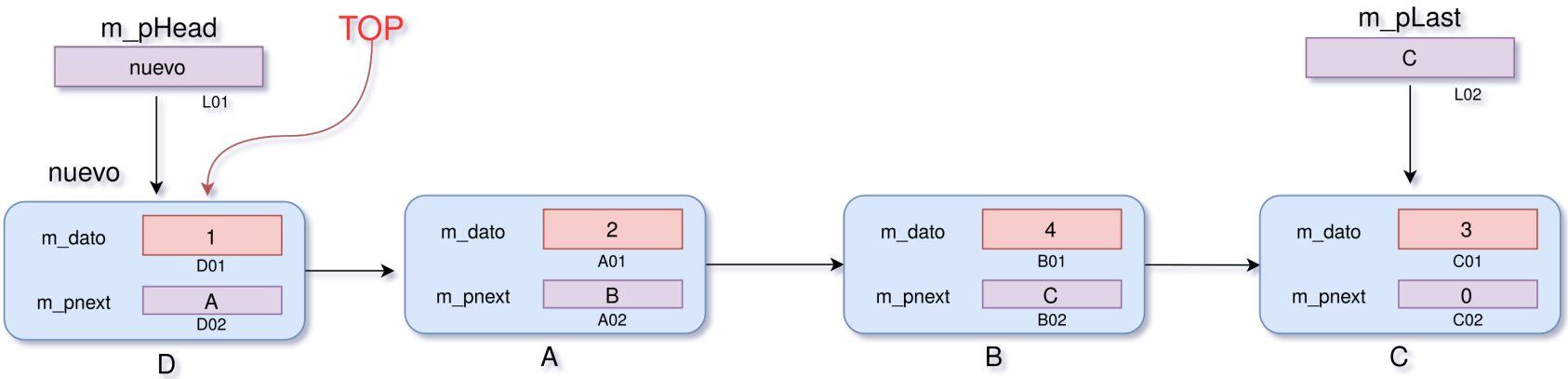


Figure : Apuntar la Cabeza al Nuevo

Push - Lista

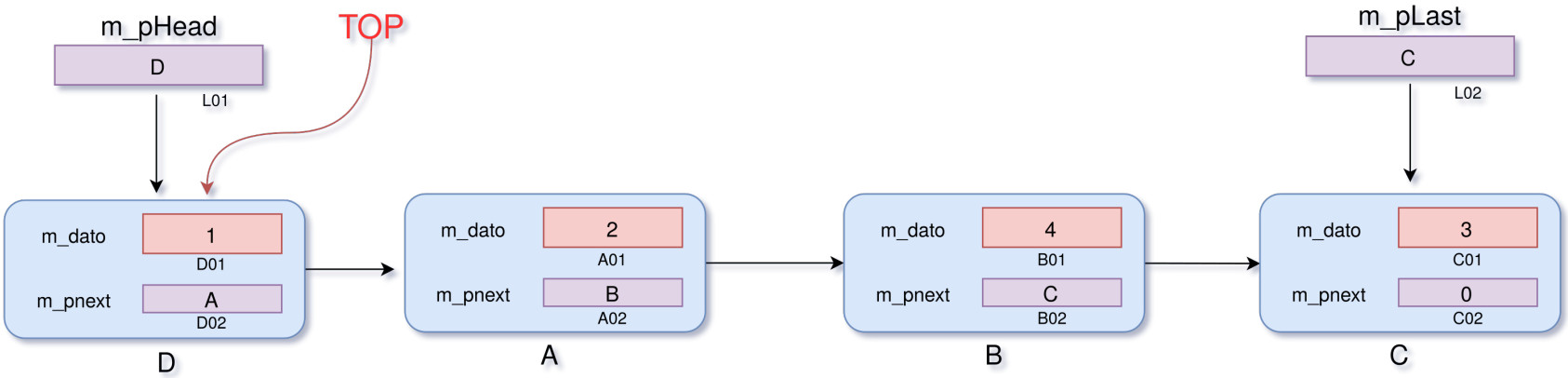


Figure : Resultado

Push

```
1  template <class T>
2  void Pila<T>::push(T d){
3    Nodo<T> * nuevo = new Nodo<T>(d);
4    if(!m_phead)
5      m_phead = nuevo;
6    else{
7      nuevo->m_psiguiente = m_phead;
8      m_phead = nuevo;
9    }
10 }
```

Pop - Lista

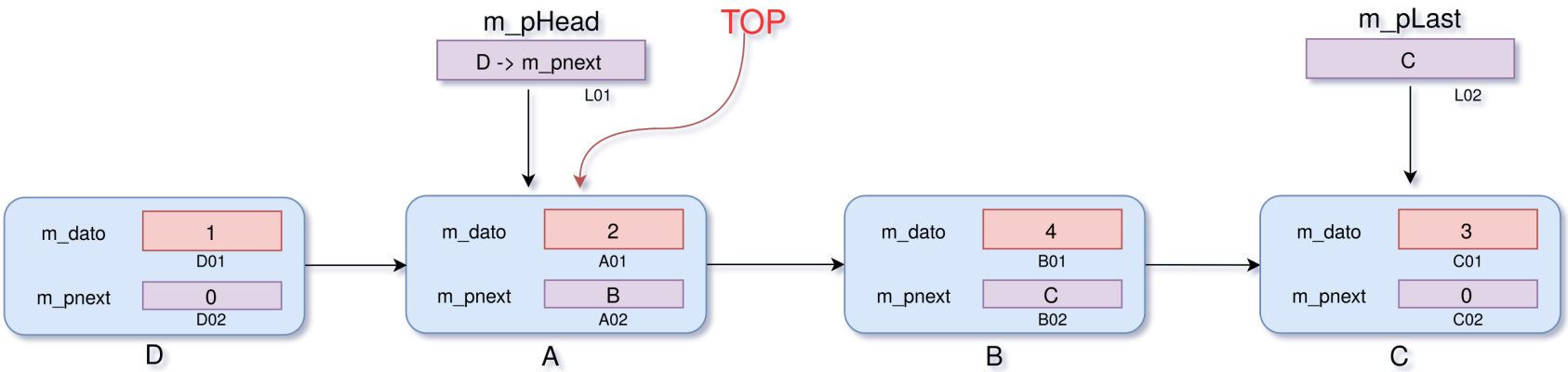


Figure : Apuntar la cabeza de la lista al siguiente de la cabeza

Pop - Lista

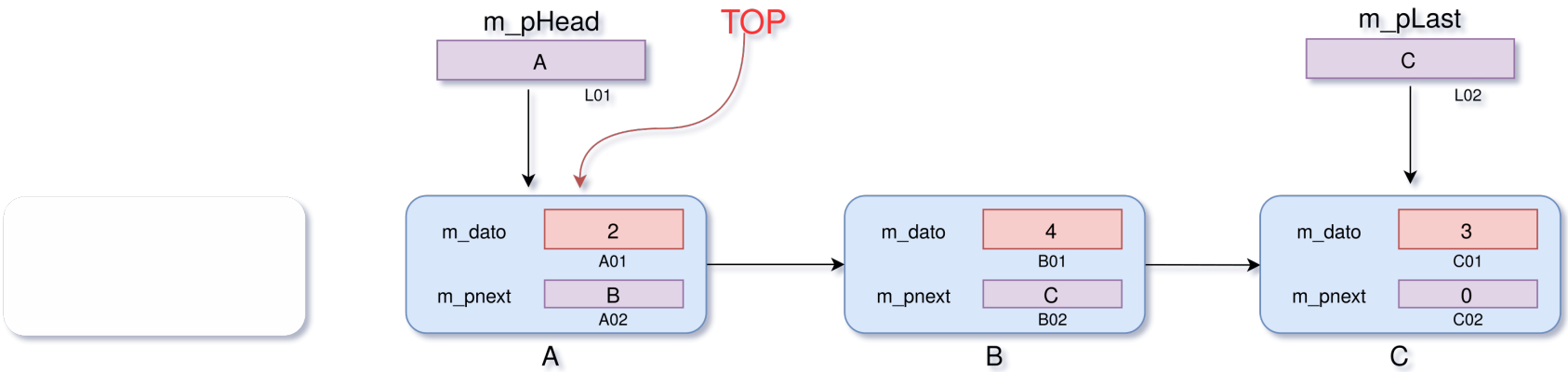


Figure : Borrar el nodo y retornar el dato

Pop

```
1  template <class T>
2  T Pila<T>::pop(){
3    Nodo<T> * temp = m_phead;
4    m_phead = m_phead->m_psiguiente;
5    T dato = temp->m_dato;
6    delete(temp);
7    return dato;
8  }
```

Print

```
1  template <class T>
2  void Pila<T>::print(ostream & os){
3    Nodo<T> * temp = m_phead;
4    while(temp){
5      os<<temp->m_dato<<"->";
6      temp = temp->m_psiguiente;
7    }
8  }
```

Introducción

Las colas son caracterizadas por ser una secuencia de elementos en la que la operación de inserción push se realiza por un extremo y la operación de extracción pop por el otro.

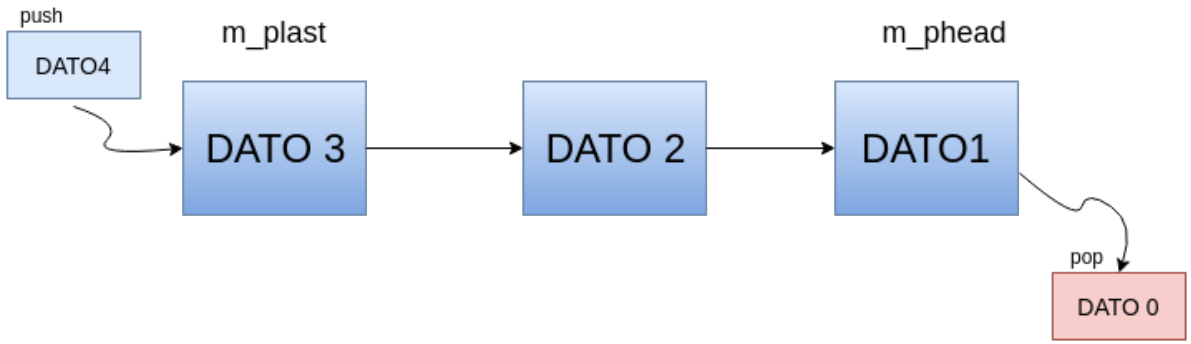


Figure : Una vista rápida

También se le llama estructura FIFO (del inglés First In First Out).

Implementacion Clase Nodo

```
1  template<class T>
2  class Nodo
3  {
4  public:
5    Nodo(T d){
6      dato = d ;
7      m_psiguiente = 0;
8    };
9    T           dato;
10   Nodo<T> *   m_psiguiente;
11 };
```

Implementacion Clase cola

```
1  #include "Nodo.h"
2  template<class T>
3  class cola
4  {public:
5    Nodo<T> * m_phead;
6    Nodo<T> * m_plast;
7  public:
8    cola(){
9      m_phead = m_plast = 0;};
10   void push(T d); // Encolar
11   T pop(); //desencolar
12   T frente(); //top
13   void imprimir();};
```

Encolar-Push

Se añade un elemento al final de esta. Si la cola esta vacía, el elemento es tanto la cabeza como la cola.

.

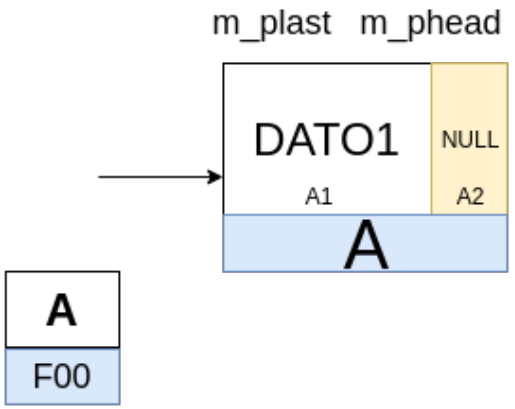


Figure : Insertar caso 1

Cuando insertamos un elemento y la cola no esta vacía, este se añade al final de la cola.
Por último actualizamos el último elemento.

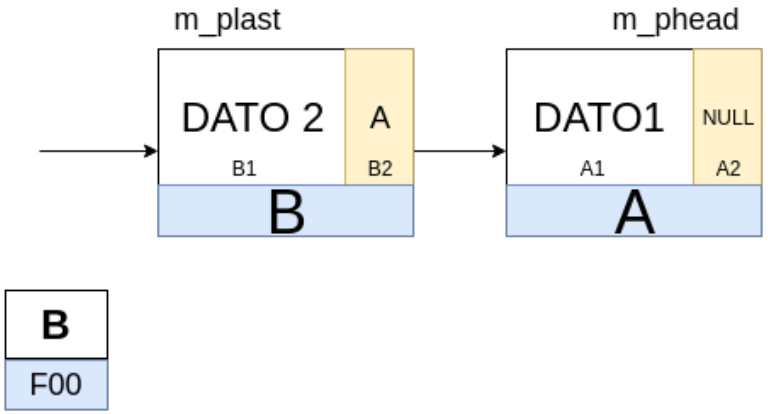


Figure : Insertar caso 2

Implementacion push

```
1  template<class T>
2  void cola<T>::push(T d){
3    Nodo<T> * nuevo = new Nodo<T>(d);
4    if (m_phead == 0){
5      m_phead = m_plast = nuevo;
6    }
7    else{
8      m_plast->m_psiguiente = nuevo;
9      m_plast = nuevo;
10   }
11 }
```

Desencolar-Pop

Se elimina el elemento frontal de la cola, el elemento que ingreso primero. .

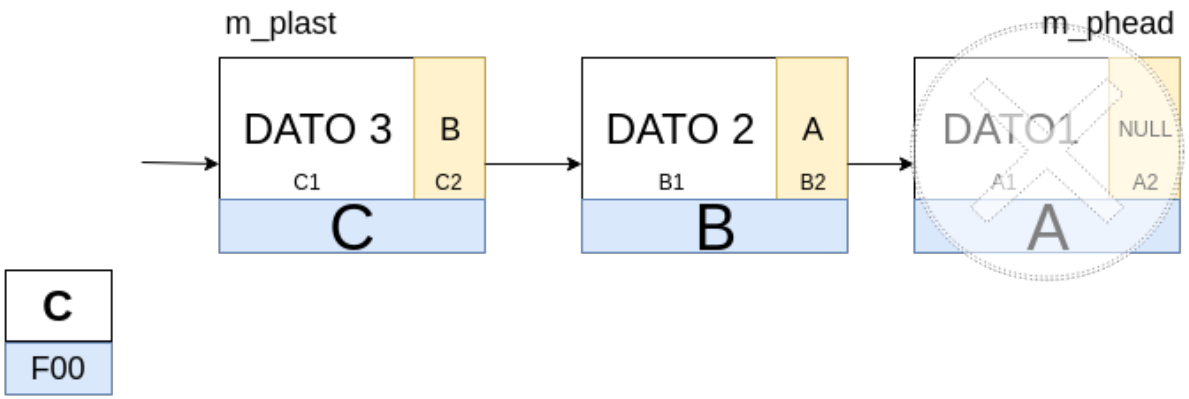


Figure : Eliminar

Desencolar-Pop

Y actualizamos el primer elemento. .

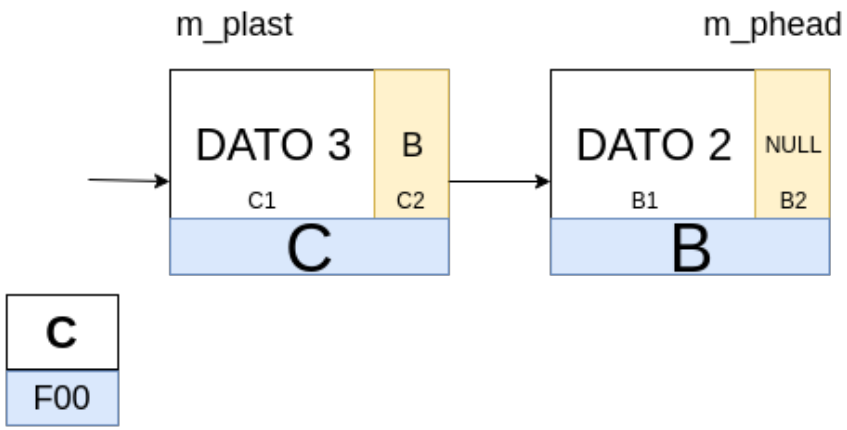


Figure : Actualizar al eliminar

Implementacion pop

```
1  template<class T>
2  T cola<T>::pop(){
3    Nodo<T> * temporal = m_phead;
4    T dato = temporal->dato;
5    m_phead = m_phead->m_psiguiente;
6    delete(temporal);
7    return dato;
8  }
```

Frente-Front

Se devuelve el elemento frontal de la cola, es decir, el primer elemento que entró.

.

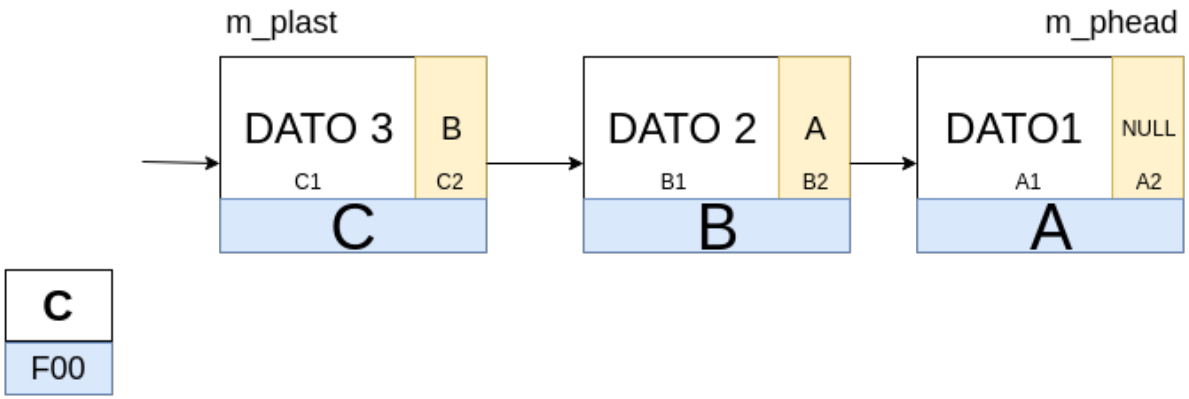


Figure : Devolver elemento frontal

Frente-Front

. .

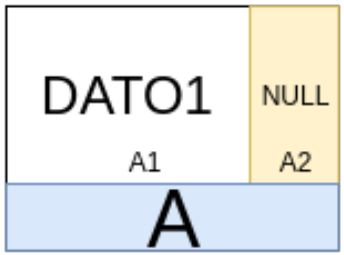


Figure : Resultado

Implementacion front

```
1  template<class T>
2  T cola<T>::frente(){
3    T dato =m_phead->dato;
4    return dato;
5  }
```

Implementacion print

```
1  template <class T>
2  void cola<T>::imprimir(){
3    Nodo<T> * temporal = m_phead;
4    if(!m_phead) return;
5    while(temporal){
6      cout<<temporal->dato<<"->";
7      temporal = temporal->m_psiguiente;
8    }
9  }
```

Bibliografía


Lopez C.[2016].*En Algoritmos y Estructuras de Datos*. Universidad Nacional de San Agustin.