Estructuras de datos lineales

Pila enlazada en C++

M.S.C. Jacob Green • 27-10-2020

Representación en memoria

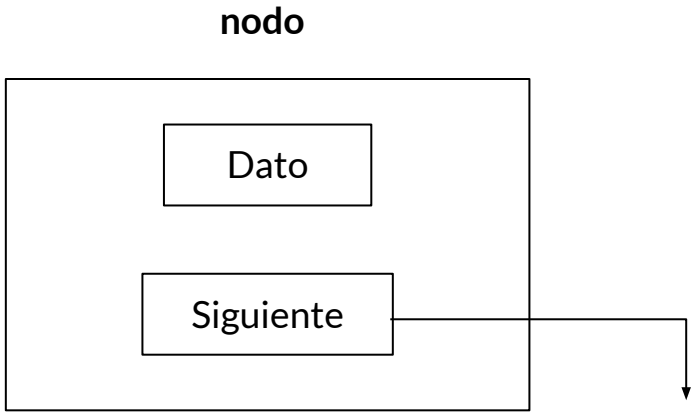
- Se pueden representar de manera enlazada, esto es, cada elemento mantiene un enlace al siguiente.
- De manera secuencial, esto es, utilizando arreglos.

Representación en memoria

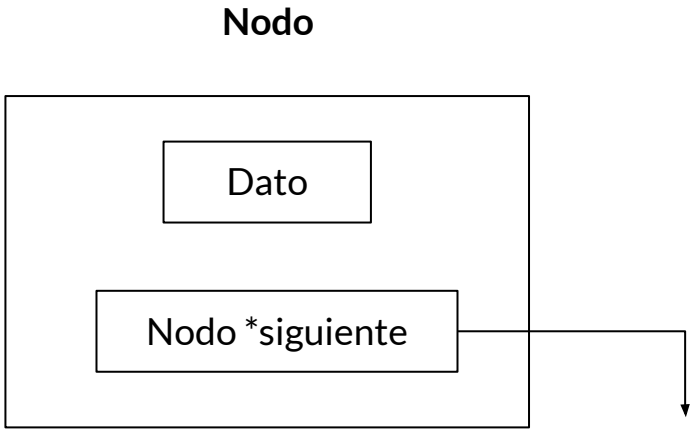
- Se pueden representar de manera enlazada, esto es, cada elemento mantiene un enlace al siguiente.

Representación gráfica de un elemento en la pila

A dicho elemento le llamaremos **nodo**



La clase Nodo



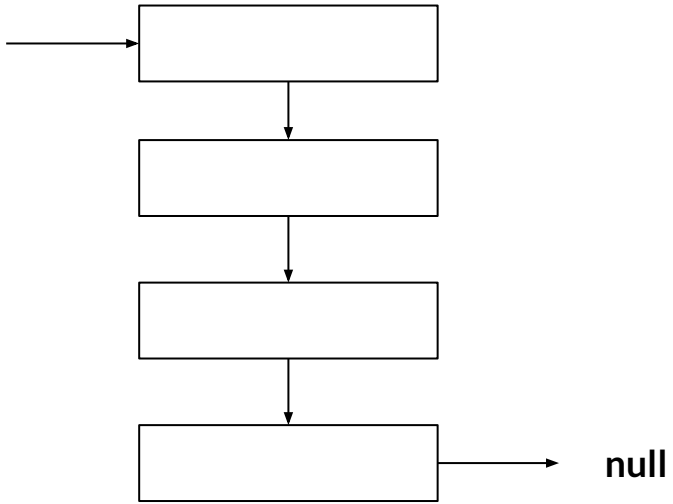

```
class Nodo {
    public:
        T dato;
        Nodo *siguiente;
        Nodo() {
        }
        Nodo(T dato, Nodo *siguiente) {
            this->dato = dato;
            this->siguiente = siguiente;
        }
};
```

Representación en memoria

- Se pueden representar de manera enlazada, esto es, cada elemento mantiene un enlace al siguiente.

Al nodo que está hasta la cima de la pila le vamos a llamar **tope**

En algunos textos también lo encontrará como **Head** o **Cabeza**



Operaciones básicas

- La operación **PUSH**, permite insertar un nodo en la cima pila.
- La operación **PEEK**, permite consultar el nodo que se encuentra en la cima.
- La operación **POP**, permite eliminar y consultar el nodo que se encuentra en la cima.

Definiendo el TDA Pila

Datos	Operaciones	Función
tope	push	Insertar un nodo encima del tope
	pop	Eliminar el nodo que se encuentra en el tope
	top	Obtener el dato que se encuentra en el tope
	empty	Verificar si la pila está vacía
	clear	Vaciar la pila
tamaño	size	Obtener la cantidad de nodos en la pila

Definiendo el TDA Pila

Datos	Operaciones	Función
Nodo *tope	void push(T d)	Insertar un nodo que **almacena el dato *d***, encima del tope
	void pop( )	Eliminar el nodo que se encuentra en el tope
	T top( )	Obtener el dato que se encuentra en el tope
	bool empty( )	Verificar si la pila está vacía
	void clear( )	Vaciar la pila
int tam	int size( )	Obtener la cantidad de nodos en la pila

El TDA Pila

```
template <class T>
class Pila {
    private:
        Nodo *tope;
        int tam;

    public:
        Pila(){
            tope = nullptr;
            tam = 0;
        }
        void push(T d);
        void pop();
        T top();
        bool empty();
        int size();
        void clear();
        ~Pila(){
            clear();
        }
};
```

Datos

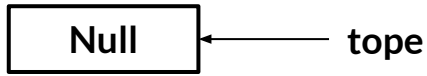
Constructor

Operaciones

Destructor

La operación push

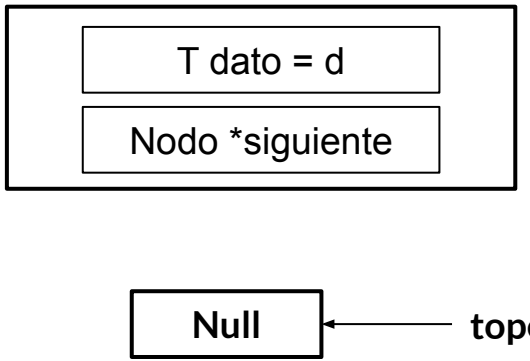
void push(T d)



La operación push

void push(T d)

Para insertar *d* en la pila, creamos un nodo que almacena dato = d

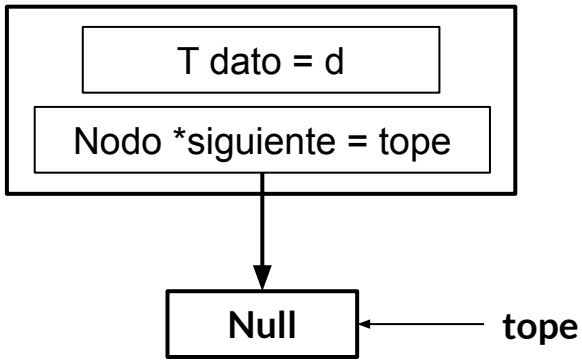


La operación push

void push(T d)

Para insertar *d* en la pila, creamos un nodo que almacena dato = d

Luego hacemos que su siguiente apunte a tope

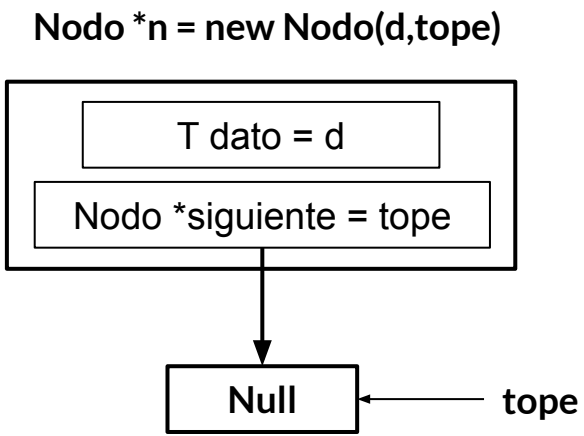


La operación push

void push(T d)

Para insertar *d* en la pila, creamos un nodo que almacena dato = d

Luego hacemos que su siguiente apunte a tope



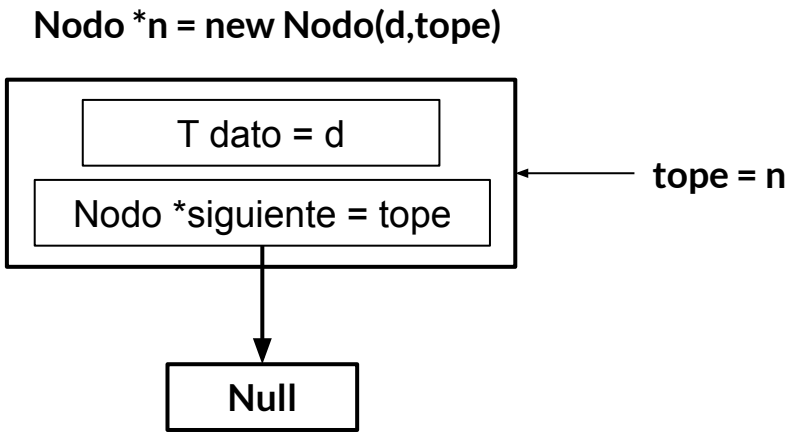
La operación push

void push(T d)

Para insertar *d* en la pila, creamos un nodo que almacena dato = d

Luego hacemos que su siguiente apunte a tope

Ahora movemos tope hacia arriba: tope = n



La operación push

void push(T d)

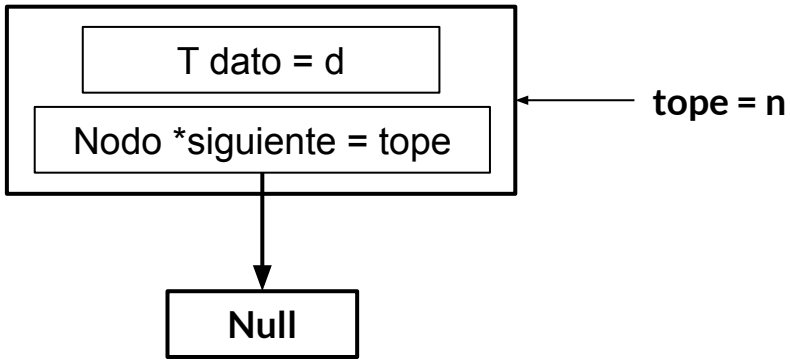
Para insertar *d* en la pila, creamos un nodo que almacena dato = d

Luego hacemos que su siguiente apunte a tope

Ahora movemos tope hacia arriba: tope = n

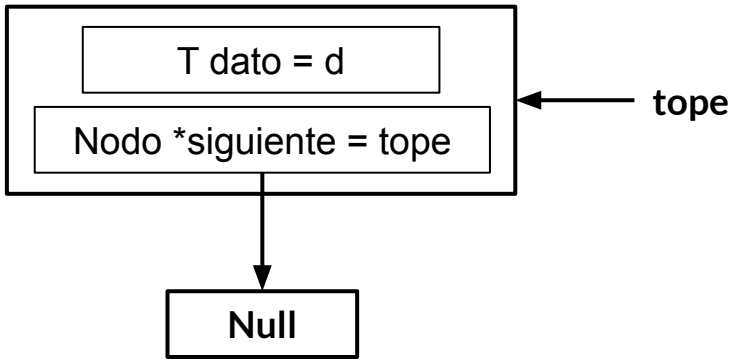
```
template <class T>
void Pila<T>::push(T d) {
    Nodo *n = new Nodo(d, tope);
    tope = n;
    tam++;
}
```

Nodo *n = new Nodo(d,tope)



La operación pop

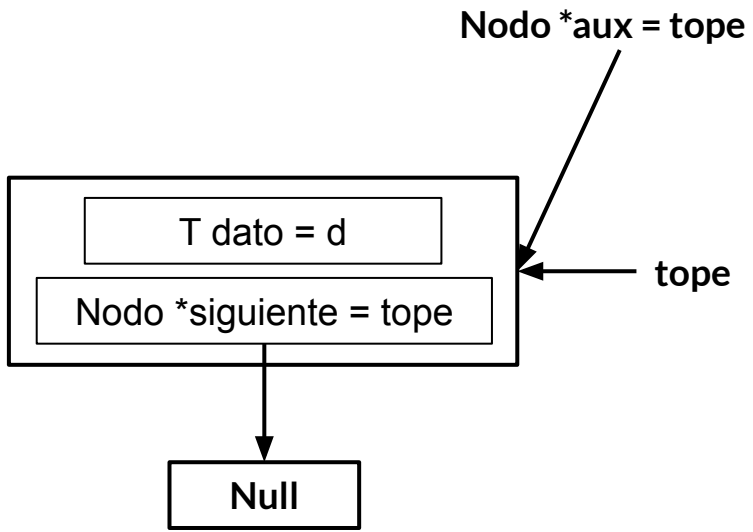
void pop( )



La operación pop

void pop( )

Para eliminar el nodo apuntado por tope, creamos un nodo auxiliar que apunte a tope

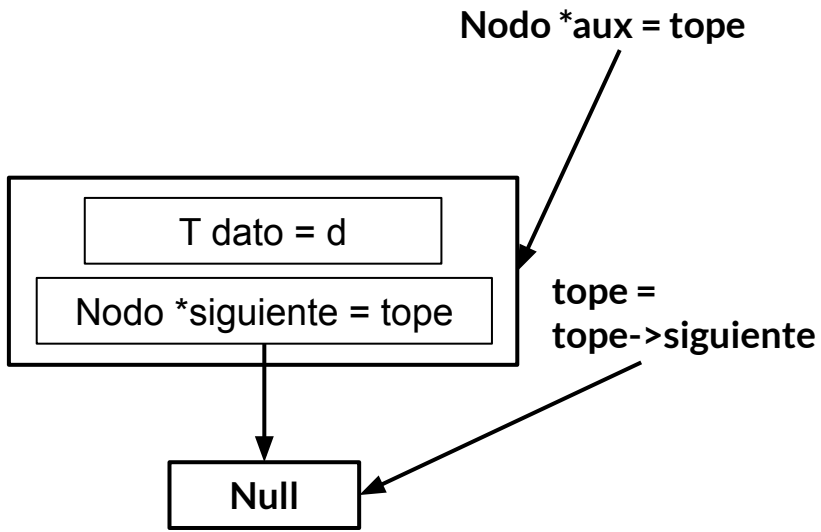


La operación pop

void pop( )

Para eliminar el nodo apuntado por tope, creamos un nodo auxiliar que apunte a tope

Luego hacemos que tope apunte a su siguiente

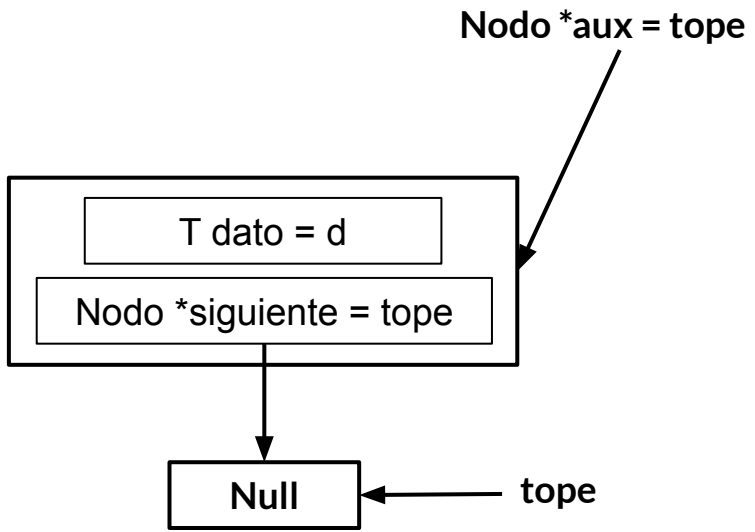


La operación pop

void pop( )

Para eliminar el nodo apuntado por tope, creamos un nodo auxiliar que apunte a tope

Luego hacemos que tope apunte a su siguiente



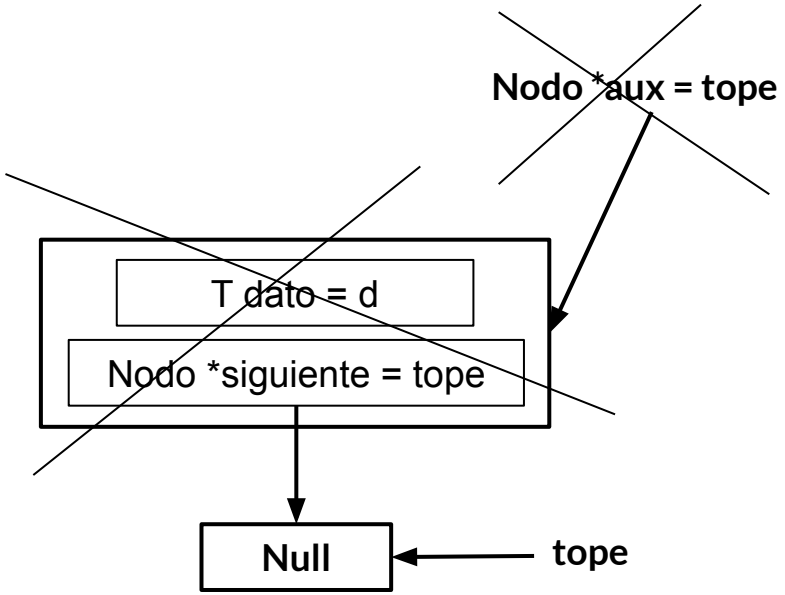
La operación pop

void pop( )

Para eliminar el nodo apuntado por tope, creamos un nodo auxiliar que apunte a tope

Luego hacemos que tope apunte a su siguiente

Ahora podemos eliminar aux:
delete aux



La operación pop

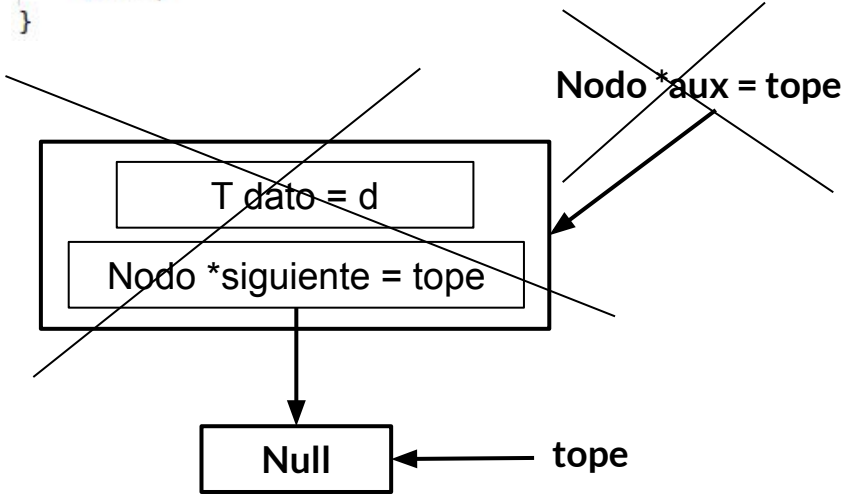
void pop( )

Para eliminar el nodo apuntado por tope, creamos un nodo auxiliar que apunte a tope

Luego hacemos que tope apunte a su siguiente

Ahora podemos eliminar aux:
delete aux

```
template <class T>
void Pila<T>::pop() {
    Nodo *aux = tope;
    tope = tope->siguiente;
    delete aux;
    tam--;
}
```



Operación top, empty y size

T top( )

```
template <class T>
T Pila<T>::top() {
    return tope->dato;
}
```

bool empty( )

```
template <class T>
bool Pila<T>::empty() {
    return tope == nullptr;
}
```

int size( )

```
template <class T>
int Pila<T>::size() {
    return tam;
}
```

La operación clear

void clear( )

```
template <class T>
void Pila<T>::clear() {
    Nodo *aux;
    while(!empty()) {
        aux = tope;
        tope = tope->siguiente;
        delete aux;
        tam--;
    }
}
```

La operación clear

void clear( )

```
template <class T>
void Pila<T>::clear() {
    Nodo *aux;
    while(!empty()) {
        aux = tope;
        tope = tope->siguiente;
        delete aux;
        tam--;
    }
}
```

Se itera sobre la pila mientras no esté vacía

En cada iteración se elimina el nodo apuntado por tope

Y luego se decrementa el tamaño de la pila

La operación clear

void clear( )

```
template <class T>
void Pila<T>::clear() {
    Nodo *aux;
    while(!empty()) {
        aux = tope;
        tope = tope->siguiente;
        delete aux;
        tam--;
    }
}
```

Se itera sobre la pila mientras no esté vacía

En cada iteración se elimina el nodo apuntado por tope

Y luego se decrementa el tamaño de la pila

Podemos verla como si se ejecutara la operación pop( ) hasta que la pila esté vacía

```
while(!empty()) {
    pop();
}
```

Utilización de nuestra clase Pila

```cpp
#include <iostream>
#include "Pila.hpp"
using namespace std;

int main() {

    Pila<int> pila;
    pila.push(1);
    pila.push(2);
    pila.push(3);

    cout<<"Dato = "<<pila.top()<<endl;
    pila.pop();
    cout<<"Dato = "<<pila.top()<<endl;
    pila.pop();
    cout<<"Dato = "<<pila.top()<<endl;
    pila.pop();

    if(!pila.empty())
        cout<<pila.top()<<endl;
    else cout<<"Pila vacia."<<endl;

    return 0;
}
```

Utilización de nuestra clase Pila

```cpp
#include <iostream>
#include "Pila.hpp"
using namespace std;

int main() {

    Pila<int> pila;
    pila.push(1);
    pila.push(2);
    pila.push(3);

    cout<<"Dato = "<<pila.top()<<endl;
    pila.pop();
    cout<<"Dato = "<<pila.top()<<endl;
    pila.pop();
    cout<<"Dato = "<<pila.top()<<endl;
    pila.pop();

    if(!pila.empty())
        cout<<pila.top()<<endl;
    else cout<<"Pila vacia."<<endl;

    return 0;
}
```

```
Dato = 3
Dato = 2
Dato = 1
Pila vacia.
```

Código

https://github.com/JGreen86/EstructurasDeDatos/tree/master/EstructurasLineales/PilaEnlazada