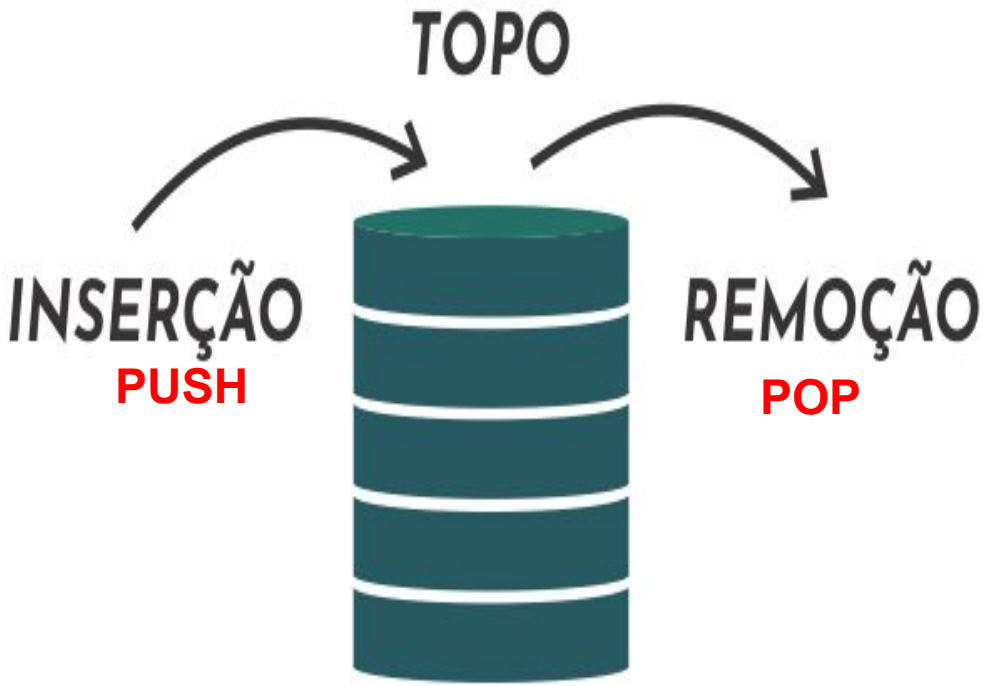
LIFO (Pilha)
Dinâmica

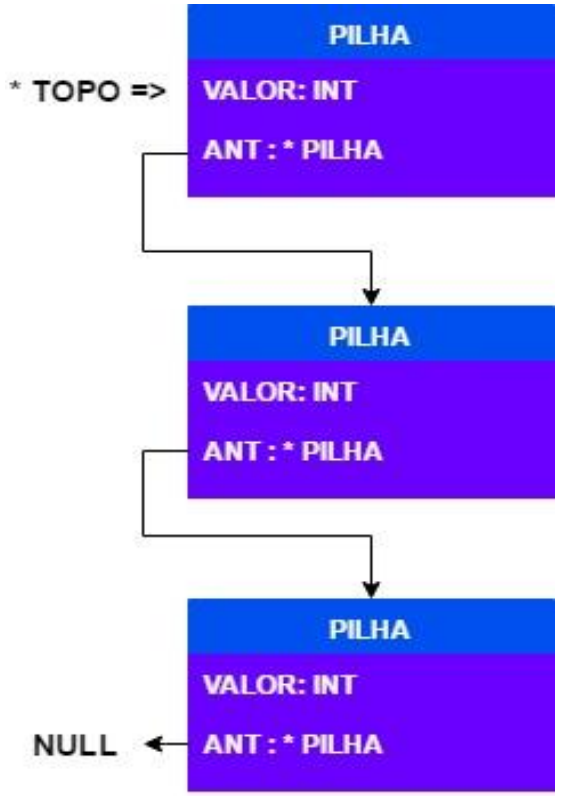
Conceito de LIFO dinâmica

LIFO => Trata-se de uma estrutura do tipo PILHA, onde o último elemento (**TOPO**) que entra será o primeiro elemento a sair. É conhecida como a estrutura do tipo **LAST-IN FIRST-OUT**, exatamente como uma pilha de pratos ou de livros qualquer. Por ser **dinâmica**, o número de elementos ou nós de uma **LIFO** dependerá da quantidade de memória **RAM** disponível para armazenamento.



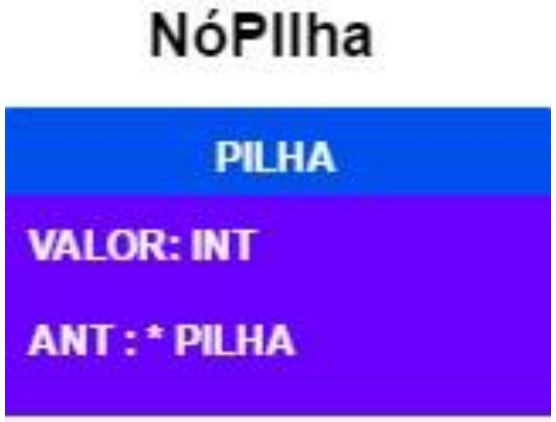
Encadeamento de uma LIFO (Pilha)

NÓ => O último elemento (nó) de uma **LIFO** chama-se Topo, sendo que deverá haver um ponteiro também chamado **TOPO** que irá armazenar o endereço deste último elemento. O Encadeamento da LIFO deverá ser feito a partir do **TOPO**, onde o ponteiro interior do tipo **PILHA**, chamado **ANTERIOR** ou **ANT** deverá apontar sempre para o nó anterior até chegar no primeiro nó da **LIFO**, quando deverá apontar para **NULL**.



Representação de um NÓ com STRUCT

Para representar um NÓ da LIFO, utilizaremos um STRUCT com um ponteiro interior chamado ANT que sempre irá armazenar o endereço do nó anterior até chegar no primeiro que irá apontar para **NULL**.



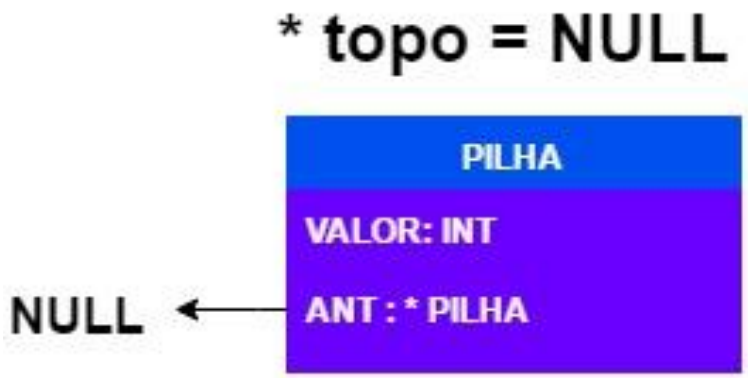

```
typedef struct Nopilha pilha;

struct Nopilha
{
    int valor;
    pilha *ant;
};

pilha * topo;
```

Código Construtor da LIFO

O código construtor de uma LIFO terá o objetivo de inicializar o ponteiro topo com NULL.




```
#include <iostream>
#include <cstdlib>
#include <stdlib.h>
using namespace std;

typedef struct nopilha pilha;
struct nopilha
{
  int valor;
  pilha *ant;
};

int cont; // para contar os nós
pilha * topo;
void construtor ( ) {
 topo = NULL;   cont = 0;          }
```

Código para Inserir Elementos na LIFO

Na função **push(),** a instrução **pilha *newpilha = (pilha*) malloc(sizeof(pilha));** cria um novo Nó chamado newpilha do tipo pilha. A instrução **if (newpilha == NULL) return false;** devolve false caso não seja possível a alocação de memória. Mas se a alocação ocorrer, o argumento (**valor)** será atributo ao atributo **valor** de newpilha: **newpilha->valor = valor;** em seguida o ponteiro interno ant irá apontar para o último topo: **newpilha->ant = topo;** finalmente, a última instrução irá armazenar o endereço de newpilha no ponteiro topo, newpilha se torna o topo.

```
bool push(int valor)
{
  pilha *newpilha = (pilha*) malloc(sizeof(pilha));

  if (newpilha == NULL) return false;

   newpilha->valor = valor;
   newpilha->ant   = topo;
             topo = newpilha;

  cont ++; // incrementa quantidade de nós
  return true;
}
```

Código para remover elementos da LIFO

A função **pop()** sempre irá excluir o topo da **pilha**, basta guardar o endereço de memória do **(*topo)** dentro de um ponteiro temporário do mesmo tipo chamado **(*temp): temp = topo;** depois guardar o valor a ser removido: **valor = topo->valor;** e finalmente tornar o elemento anterior no novo **topo**: **topo = topo->ant;** Para finalizar utilizamos a função free para tirar o endereço do temp, isto é, do topo excluído da memória. **free(temp);**

```
bool pop()
{
  int valor;   pilha *temp;
  temp = topo;
  valor = topo->valor;
  topo = topo->ant;

  free(temp);
  cout << "\nO valor removido foi:" << valor << endl;
  system("sleep 3");
  return true;
}
```

Código para exibir a LIFO

Para exibir a LIFO, basta usar um ponteiro temporário e percorrer a estrutura exibindo cada elemento até chegar no último elemento que terá NULL como endereço.

```
void exibirpilha()
{
  pilha *temp;
  temp= topo;
  system("clear");
  while ( temp  !=  NULL)
  {
    cout << "\n" << temp->valor << endl;
    temp = temp->ant;
  }
  system("sleep 3");
}
```

Código para verificar se a LIFO está vazia

Quando o * TOPO estiver apontando para NULL, então a LIFO está vazia, ainda não possui elementos.

```
bool vazia()
{
    return ( topo == NULL );
}
```

Código Destrutor da LIFO

Ao final do processamento será necessário fazer a liberação da memória dinâmica que foi alocada com o comando malloc(). Para tanto, a estrutura deverá ser percorrida e cada elemento **(nopilha)** armazenado em ponteiros deverá ser banido da memória usando a função free(). O último ponteiro banido deverá ser o topo.

```
void destrutor() {
  pilha *temp;

  while ( topo != NULL )
     {   temp = topo;
         topo = topo->ant;
         free ( temp );   }

  free ( topo );
  cout << "\PIlha destruida com sucesso!\n";
  system("sleep 3");  }
```

Código que retorna o total de elementos da FIFO

Basta percorrer a LIFO e contar os elementos através de uma variável contadora.

```
int total ()    {
 pilha * temp;
 temp = topo;
 int cont = 0;
 while ( temp != NULL )
   {
      cont ++;
      temp=temp->ant;
   }
 return cont;  }
```

ADO Avaliação Contínua

1. Monte e execute o programa LIFO alterando o código para C. Crie uma subrotina controle() contendo um menu para executar o código.

2. Faça alterações no programa LIFO, crie o atributo nome dentro do struct, crie uma função para buscar um valor na LIFO, quando o valor for encontrado você deverá exibir o nome que está na mesma posição do valor encontrado, você pode criar o atributo posição dentro do struct para ajudar.