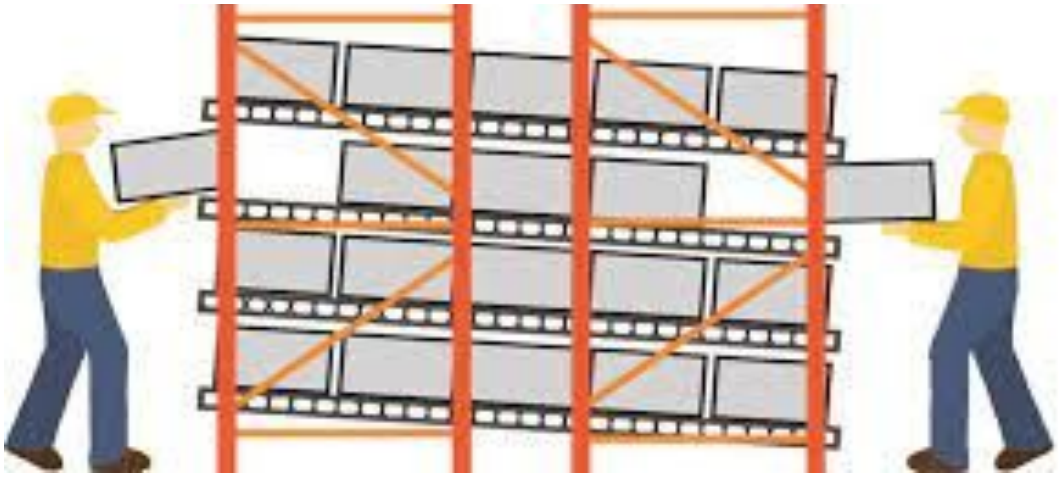
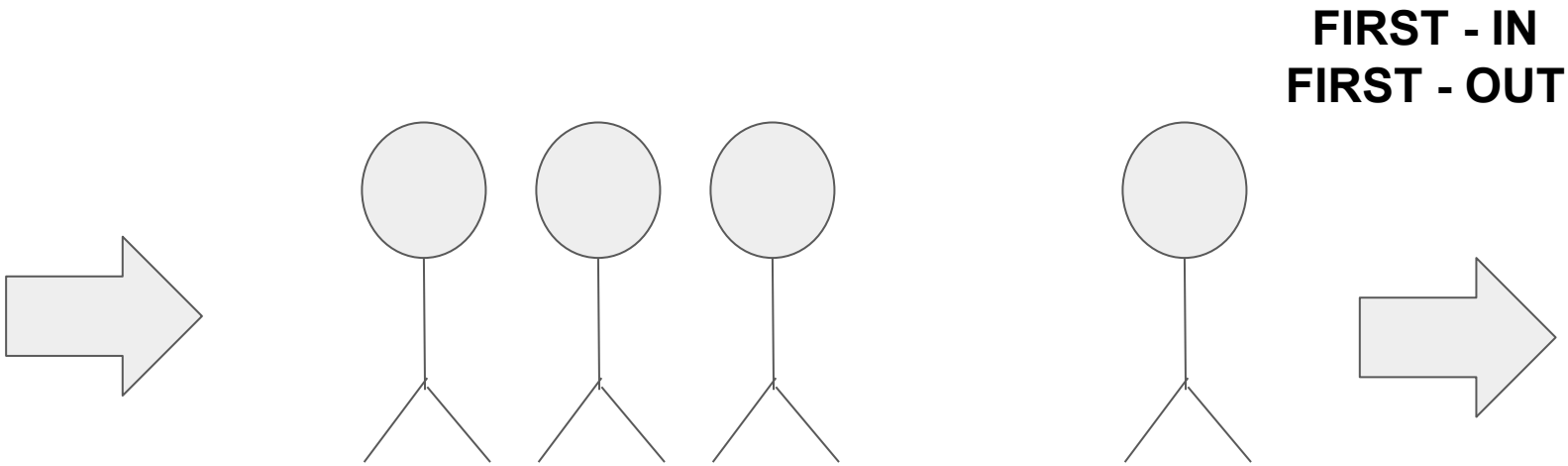
# FIFO
# ( dinâmica )
# Fila linear não circular com encadeamento simples



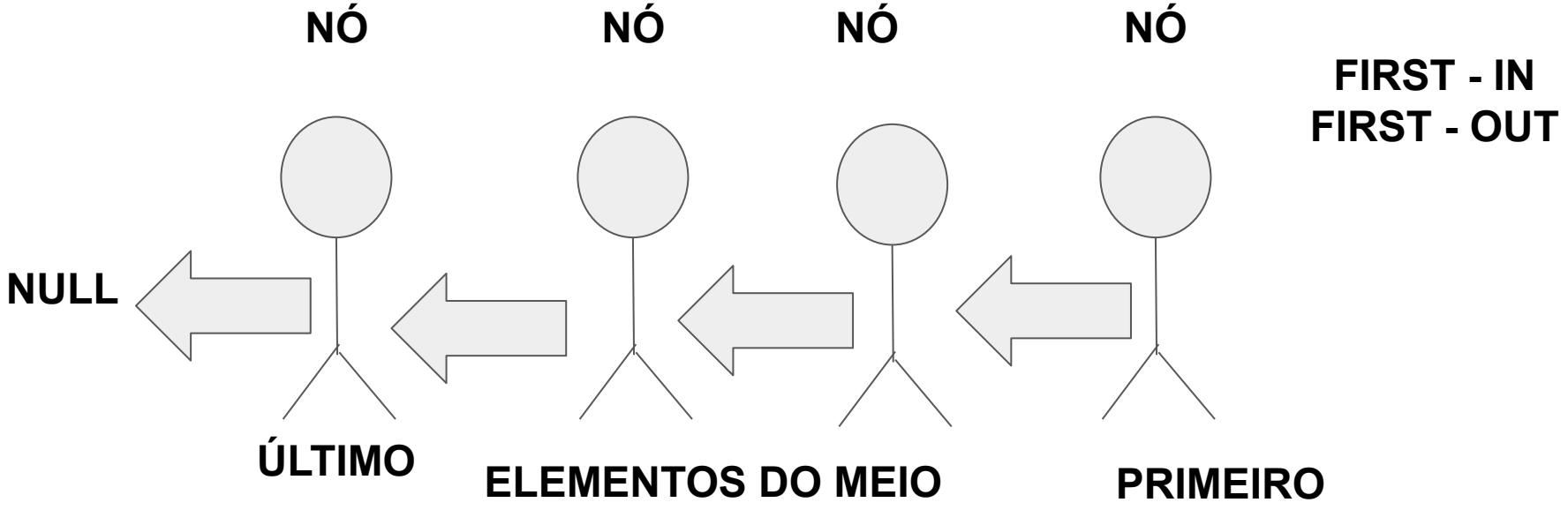
# Conceito de FIFO

FIFO => Trata-se de uma estrutura do tipo fila, onde o primeiro elemento que entra é exatamente o primeiro que irá sair. É conhecida como a estrutura do tipo FIRST- IN FIRST- OUT, exatamente como uma fila qualquer, por exemplo uma fila de banco:



# Conceito de NÓ

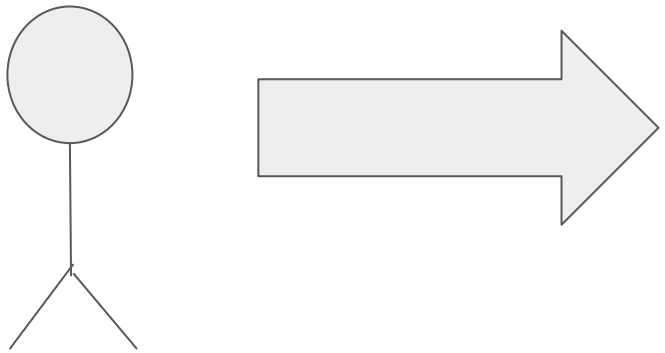
NÓ=> Cada elemento de uma FILA DINÂMICA chama-se nó, que será representado por um PONTEIRO é terá também um ponteiro interior do mesmo tipo, necessário para guardar o endereço do elemento anterior desde o PRIMEIRO até o ÚLTIMO.



# Representação de um NÓ com STRUCT

Para representar um NÓ usamos um STRUCT com um ponteiro interior que irá armazenar sempre o endereço do NÓ anterior até chegar no INÍCIO da FIFO.

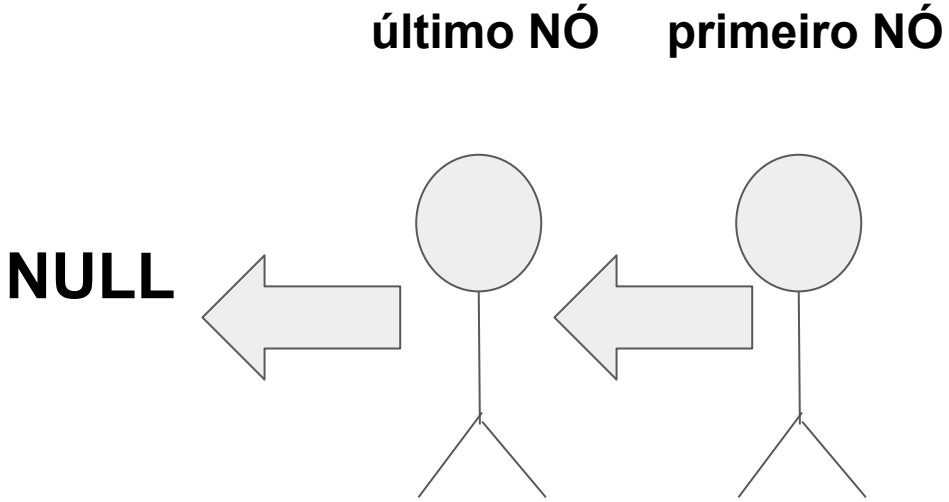
**NÓ**



```
struct MeuNo
{
   int numero;
   MeuNo * prox;
 };
```

# Código Construtor da FIFO - FIFO Vazia

O código construtor de uma FIFO terá o objetivo de inicializar as variáveis de Ponteiros ( primeiroNó e ultimoNó) que deverão ser inicializados com valor Nulo.




```
#include <iostream>
#include <cstdlib>
using namespace std;

int total;  // contar o total de
elementos
struct noFifo
{
 int chave;      // Sequencial número
 noFifo * prox; //ponteiro para guardar
o próximo elemento da fifo
};

noFifo * primeiroNo;// endereço do
primeiro elemento
noFifo * ultimoNo; // endereço do último
elemento

void construtor ()
{ primeiroNo = ultimoNo = NULL;
 total = 0;
 cout << "\nFIFO iniciada!\n";
 system("sleep 2"); }
```

# Código para verificar se a FIFO está vazia

Quando o primeiro Nó estiver apontando para NULL, então a FIFO está vazia, ainda não possui elementos.

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

# Código para Inserir Elementos na FIFO

A função malloc ( ) cria um novo Nó, deve-se atribuir o valor da CHAVE e do ponteiro interno do novo Nó aponta para NULL. Caso seja o primeiro elemento inserido, os ponteiros primeiroNo, ultimoNó e novoNo terão o mesmo endereço de memória, o PROX interno de cada um irá apontar para NULL. Caso contrário, o PROX do último Nó sempre apontará para o novo Nó. Ou seja, o novoNó será sempre o último Nó da FIFO.

```
void inserirNo ( int valor ) { noFifo * novoNo = ( noFifo * ) malloc ( sizeof (
noFifo ) );

if ( novoNo == NULL )
{ cout << "\nErro de alocação!" << endl;
   system("sleep 2");
   return ; }

// Configurar os atributos internos
   novoNo->chave = valor; // coloca o valor
   novoNo->prox  = NULL;   // configurar o ponteiro interno
   total ++;  // incrementa o total
// configura atributos internos do novo Nó
if (  vazia () == true )
   { primeiroNo = novoNo;
     ultimoNo   = novoNo;  }
else
{  ultimoNo->prox = novoNo; }

ultimoNo = novoNo;}
```

# Código para remover elementos da FIFO

Para excluir o primeiro elemento da fila, basta guardar o endereço de memória do primeiro elemento da dentro de um ponteiro temporário (**temp),** tomar o endereço do segundo elemento da fila (temp->prox) e colocar no ponteiro primeiro Nó, finalmente excluir o antigo endereço de memória do primeiro Nó com a função free( ).

```
void removerNo()
{
    noFifo  * temp; // ponteiro temporário
    temp = primeiroNo; // guarda o primeiro Nó que será removido
       if (  ! vazia ( )  )
       {
              primeiroNo = temp->prox;
       }

       free ( temp );
       total --; // decrementa o total
}
```

# Código para exibir a fila FIFO

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

```
void mostrar ()
{
        noFifo * temp;
        temp = primeiroNo;

 while ( temp != NULL )
 {
    cout << temp->chave << " ";
    temp = temp->prox;
 }

   cout << endl;
   system("sleep 3");
free(temp); }
```

# Código Destrutor da FIFO

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 é cada elemento armazenado em ponteiro deverá ser banido da memória com o comando free().

```
void destrutor()
{
        noFifo * temp, * aux;
        temp = primeiroNo;
        while ( temp != NULL )
              {
                    aux = temp;
                    temp = temp->prox;
                    free ( aux );
              }

        cout << "\nFIFO destruída...!" << endl;
        system("sleep 3");
}
```

# ADO Avaliação Contínua

1. Monte o programa FIFO colocando um menu de controle;
2. Altere o código para C;
3. Altere o código da FIFO, acrescente os atributos Nome e Cpf, faça as funções para buscar o nome de uma pessoa e também para excluir um registro através do nome, se você excluir um elemento da fila então deverá redirecionar os ponteiros para não quebrar o encadeamento da fila.