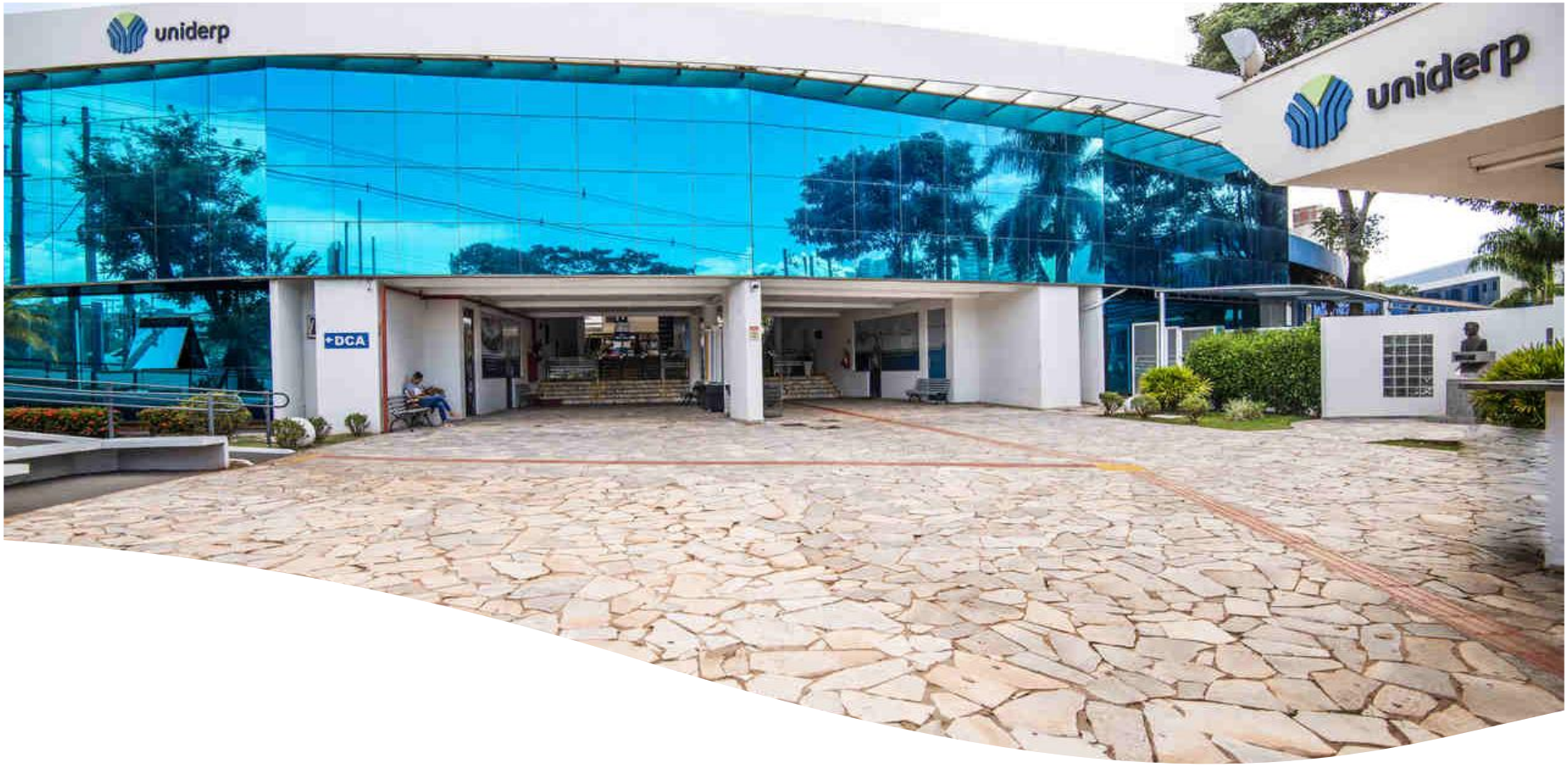


Algoritmos e Estrutura de Dados Avançado

Encontro 02 – Ponteiros

uniderp

Materiais e dúvidas

Nenhum material será enviado via e-mail.

Os materiais serão disponibilizados no **AVA** e no disco virtual: **bit.ly/edauniderp**

Dúvidas, questionamentos, entre outros deverão ser realizados **APENAS** pelo **e-mail** e **AVA.**

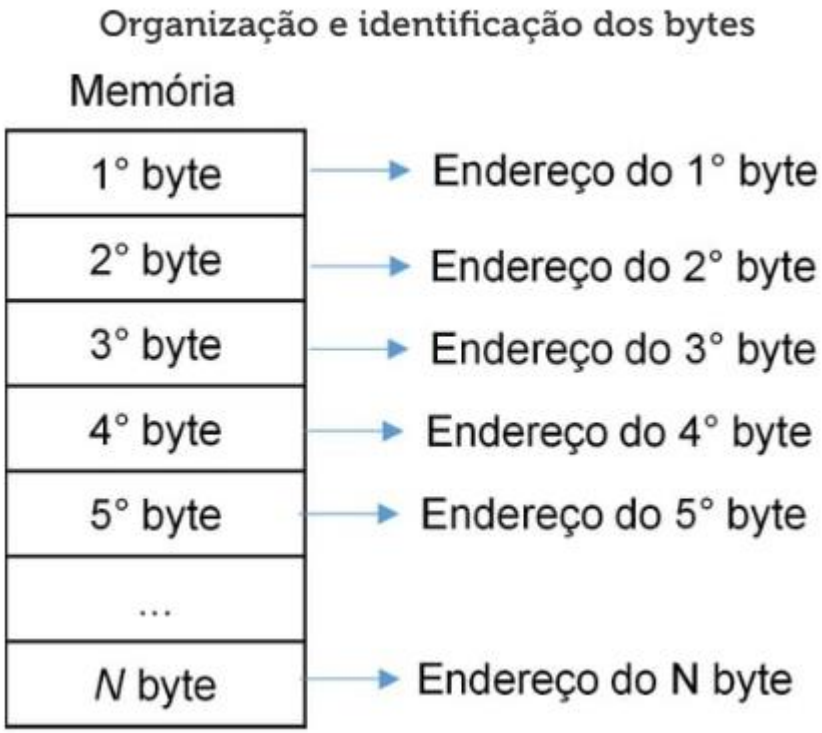
Para ingressar no grupo do **Whatsapp** da disciplina acesse o link **linklist.bio/noiza** e selecione sua disciplina.

Ponteiros

- Muitos sistemas operacionais, compiladores e linguagens de programação são construídos inteiramente ou parcialmente na linguagem C, e isso só é possível graças aos ponteiros.
- Para entendermos ponteiros devemos antes compreender como funciona a memória de trabalho do computador, ou simplesmente memória RAM.

Memória RAM

- A memória RAM é o local utilizado pelos softwares para armazenamento temporário de dados, sendo organizada como uma sequência de bytes.
- A identificação de cada byte na memória ocorre pelo seu endereço, conforme ilustra a figura.



Alocação de Ponteiros

- Cada constante ou variável na memória ocupa um espaço que pode variar de 1 a *N bytes*.
- Por exemplo:
  - uma variável do tipo *char* ocupa 1 *byte*;
  - uma variável do tipo inteira (*int*) ocupa 4 *bytes*;
  - uma variável do tipo *double* ocupa 8 *bytes*.
- Cada variável alocada está relacionada ao endereço do *byte* na memória.

Tabela de tipos inteiros

Tipo	Num de bits	Formato para leitura com scanf	Intervalo
char	8	%c	De -128 até 127
*int	16	%d	De -32.768 até 32.767
int	32	%d	De -2.147.483.648 até 2.147.483.647
long int	32	%li	De -2.147.483.648 até 2.147.483.647
float	32	%f	De 3,4E-38 até 3.4E+38
double	64	%lf	De 1,7E-308 até 1,7E+308
bool	1	%i	true ou false

- Para arquiteturas 32 bits o tipo int armazena 16 bits.
- O tipo double usa 8 bytes (53 bits para a mantissa e 11 para o expoente);
- O tipo float usa 4 bytes (24 bits para a mantissa e 8 para o expoente).

Alocação de Ponteiros

- Caso a variável ocupe mais que 1 *byte*, o endereço será do primeiro *byte*.

Por exemplo:

- Ao alocarmos a variável *int* x, ela ocupará uma sequência de 4 *bytes*, entretanto o endereço usado será o do primeiro *byte*.

Alocação de Ponteiros

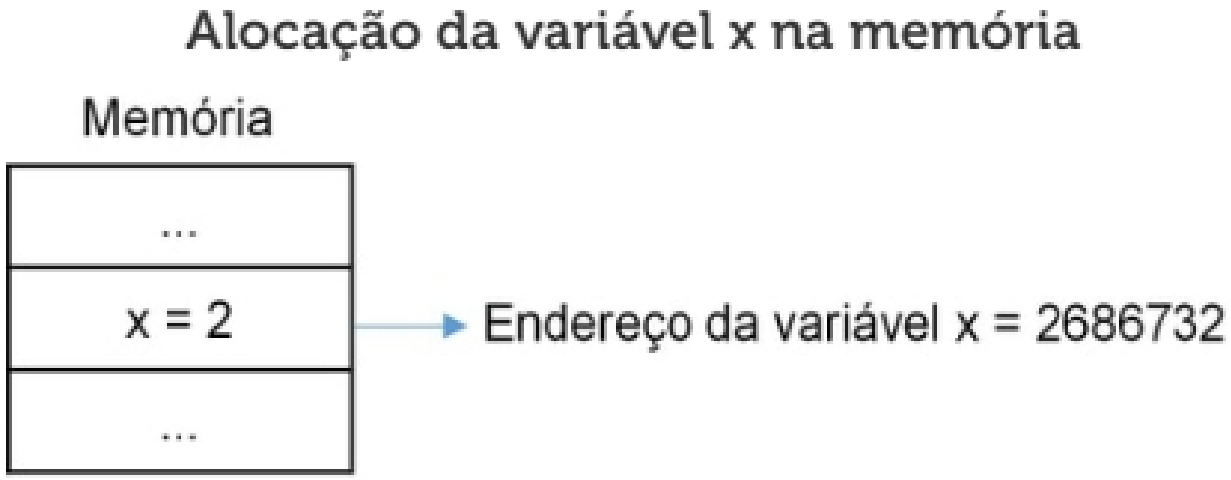
- Na linguagem C utilizamos o operador (&) para conhecer e acessar o endereço de uma variável na memória. Veja um exemplo de sua utilização, com seu respectivo resultado (exemplo01):

```
Valor de x = 2
Endereco de x = 2686732
```

```c
#include <stdio.h>
int main (){
    int x = 2;
    printf("Valor de x = %d\n", x);
    printf("Endereco de x = %d", &x);
    return 0;
}
```

Alocação de Ponteiros

- No exemplo anterior, a linha 4 imprime o valor guardado na variável x, ou seja, 2. Já a linha 5 imprime um valor inteiro referente ao endereço, do primeiro *byte* da variável x na memória (lembrando que uma variável inteira ocupa 4 *bytes*).
- A figura, a seguir, ilustra a organização na memória desse exemplo:



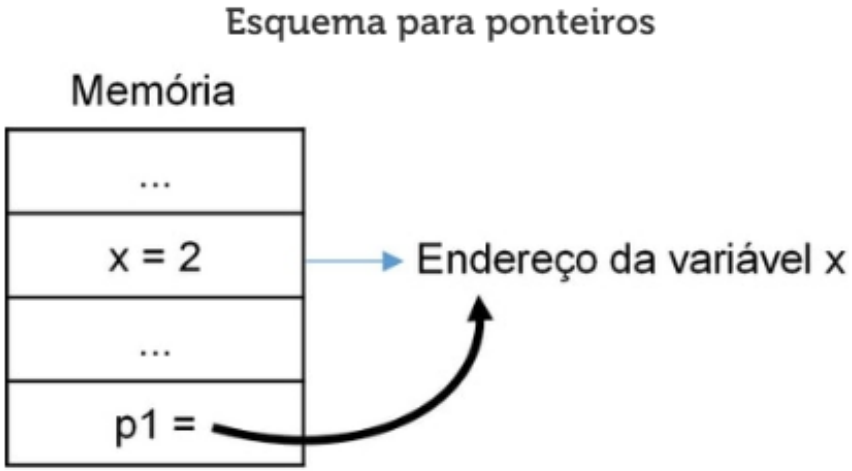
Definição de ponteiros

- Agora que sabemos que a memória se divide em pequenas partes e que cada parte está associada a um endereço, podemos entender o que é um ponteiro:

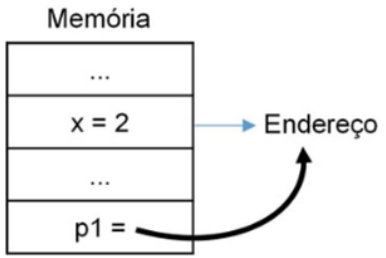
Ponteiro, por definição, é um tipo especial de variável que armazena endereços (MANZANO, 2015).

Definição de ponteiros

- Quando o ponteiro armazena o endereço de uma variável, dizemos que ele "aponta" para uma variável. Por exemplo, se o ponteiro *p1* armazena o endereço da variável x, dizemos que "*p1 aponta para x*". De modo mais formal, é o mesmo que afirmarmos que "*p1 é uma referência à variável x*", conforme ilustra a figura a seguir:



Definição de ponteiros



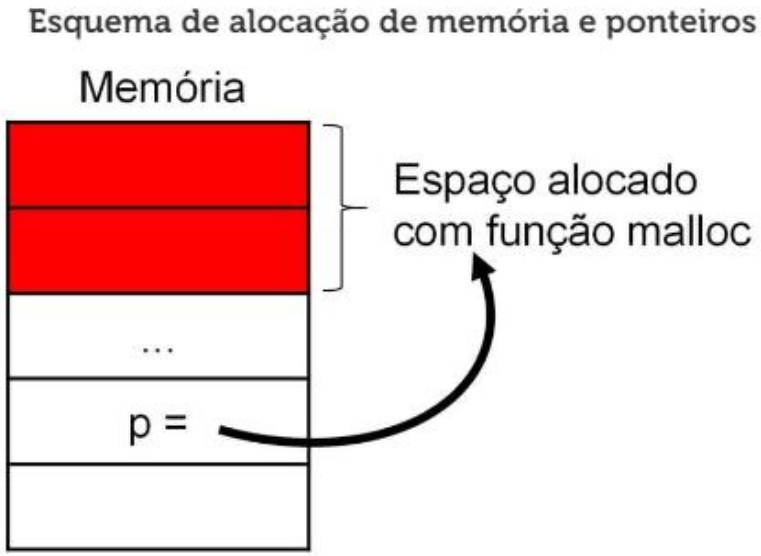
- A partir do esquema para ponteiros apresentado, podemos fazer algumas observações:
  - p1 é uma variável (especial), portanto também ocupa um espaço na memória;
  - p1 contém o endereço da variável que ele aponta, no caso, o endereço de x;
  - Como p1 contém o endereço da variável, ele pode acessar o conteúdo dessa variável tanto para leitura como para escrita;
  - Há vários tipos de ponteiros: ponteiros para inteiros, ponteiros para bytes, ponteiros para ponteiros, ponteiros para registros, ponteiros para ponteiros para ponteiros, etc. O tipo de ponteiro depende da variável que ele aponta.

Cenário para utilização de ponteiros

- Os ponteiros podem ser utilizados em qualquer situação, pois, por definição, também são variáveis. Porém, existem duas situações em que sua utilização se faz necessária:
  - Para alocar memória dinâmica (funções malloc, calloc, etc.).
  - Para fazer referências a listas, filas, enfim, estruturas de dados.

Cenário para utilização de ponteiros

- Em ambos os casos, a justificativa está no fato de serem usadas funções para reservar determinado espaço na memória.
- Para acessar esse espaço é necessária uma variável que aponte para o local, conforme ilustra a figura a seguir:



Cenário para utilização de ponteiros

- Como há vários tipos de ponteiros, o computador precisa identificar de que tipo de ponteiro nós precisamos.
- Criar um ponteiro na linguagem C é semelhante a criar uma variável qualquer, portanto precisamos informar o tipo e o nome.
- O nome da variável deve ser precedido de um asterisco (*) para identificar que a variável é um ponteiro.
- Sintaxe de declaração de ponteiro: ***tipo *nome*** ou, ainda, ***tipo* nome***.
- Exemplos:

```
int *p1;
float *p2;
char *p3;
```

Cenário para utilização de ponteiros

- Após a declaração, precisamos especificar para onde esse ponteiro aponta, que poderá ser uma variável específica ou uma região da memória alocada por funções específicas. Vejamos um exemplo:

```
1. float x;
2. float *ptr;
3. ptr = &x;
```

- Na linha 1 é declarada uma variável x do tipo *float*.
- O ponteiro na linha 2 também é declarado *float*, o ponteiro deve ser do mesmo tipo da variável para a qual ele apontará.
- Na linha 3 é feita a ligação entre o ponteiro e a variável x. Observe que foi usado &x para armazenar em *ptr* o endereço da variável x.

Cenário para utilização de ponteiros

- Vamos implementar em C o uso do ponteiro *ptr* (exemplo02).

```c
#include <stdio.h>
int main (){
    int x = 2;
    int *ptr;
    ptr = &x;

    printf("Valor de x = %d\n", x);
    printf("Endereco de x = %d\n", &x);
    printf("Conteudo armazenado no ponteiro = %d\n", ptr);
    printf("Conteudo do local onde o ponteiro aponta = %d\n", *ptr);
    return 0;
}
```

```
Valor de x = 2
Endereco de x = 2686728
Conteudo armazenado no ponteiro = 2686728
Conteudo do local onde o ponteiro aponta = 2
```

Considerações sobre Ponteiro

- O uso de ponteiro é indicado para os casos de alocação dinâmica de memória.
- A declaração de um ponteiro na linguagem C é similar à de outra variável, ou seja, especifica-se o tipo e o nome (precedido por asterisco).
- Exemplo:

```
int *ptr;
```

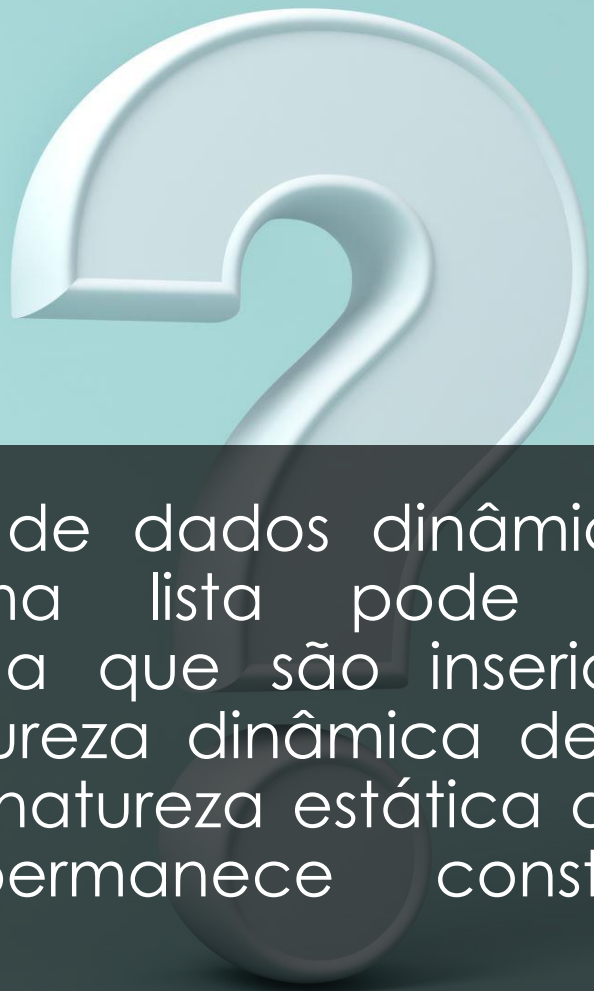
Considerações sobre Ponteiro

- A ligação do ponteiro com o endereço na memória deve ser feita usando o operador &.
- Exemplo: ***ptr = &x.***
- O valor armazenado no ponteiro é o endereço do local para o qual ele aponta.
- Exemplo: ***printf("Armazenado = %d", ptr);***
- Através do ponteiro é possível acessar o conteúdo do local da memória para o qual ele aponta.
- Exemplo: ***printf("Conteudo = %d", *ptr)***

Listas Ligadas

- Uma **Lista Ligada**, também conhecida como **Lista Encadeada**, é um conjunto de dados dispostos por uma sequência de nós, cuja relação de sucessão desses elementos é determinada por um ponteiro que indica a posição do próximo elemento, podendo estar ordenado ou não (Silva, 2007).
- Uma lista é uma coleção de L: $[a_1, a_2, ..., a_n]$ em que $n>0$ e sua propriedade estrutural baseia-se apenas na posição relativa dos elementos, que são dispostos linearmente.

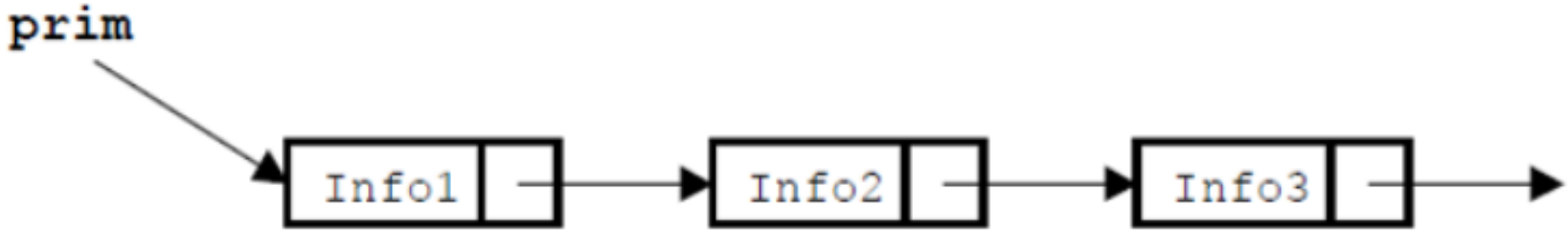
Listas Ligadas



"Uma lista é uma estrutura de dados dinâmica. O número de nós de uma lista pode variar consideravelmente à medida que são inseridos e removidos elementos. A natureza dinâmica de uma lista pode ser comparada à natureza estática de um vetor, cujo tamanho permanece constante" (Tenenbaum, 2007 p. 225).

Listas Ligadas

- O nó de lista é composto por duas informações, uma informação armazenada e um ponteiro que indica o próximo elemento da lista.



Fonte: Celes, W., Cerqueira, R., Rangel, J. L. (2004, p. 135).

Listas Ligadas

- Diferente dos vetores, em que o armazenamento é realizado de forma contígua, a Lista Ligada estabelece a sequência de forma lógica.
- Na lista encadeada é definido um ponto inicial ou um ponteiro para o começo da lista, e a partir daí pode se inserir elementos, remover ou realizar buscas na lista.
- Quando uma lista está sem nós, é então definida como lista vazia ou lista nula, assim o valor do ponteiro para o próximo nó da lista é considerado como ponteiro nulo.

Listas Ligadas

- Procedimentos básicos de manipulação de listas:

Criação ou definição da estrutura de uma lista	Inicialização da lista	Inserção com base em um endereço como referência
Alocação de um endereço de nó para inserção na lista	Remoção do nó com base em um endereço como referência	Deslocamento do nó removido da lista

Criação ou definição da estrutura da lista

- Ao criar uma estrutura de uma lista, definimos também o tipo de dados que será utilizado em sua implementação.
  Os elementos de informação de uma lista podem ser do tipo int, char e/ou float.

```
struct Lista {
    int num;
    struct Lista* prox;
}
typedef struct Lista node;
```

Inicialização da lista

- Precisamos inicializar a lista para a utilizarmos após sua criação.
- Para isso, uma das formas possíveis é criar uma função em que inicializamos a lista como nula.

```
void inicia_lista (node* LISTA){
    LISTA->prox = NULL;
}
```

Lista Ligada

- Em uma lista, precisamos alocar o tipo de dado no qual foram declarados os elementos dela, e por esse tipo de dados ocupar vários bytes na memória, precisaremos utilizar a função ***sizeof***, que permite nos informar quantos bytes o tipo de elemento criado terá (exemplo03).

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main (){
4      int *p;
5      p = (int *) malloc(sizeof(int));
6
7      if (!p)
8          printf("Erro de memoria insuficiente!");
9      else
10         printf("Memoria alocada com sucesso!");
11     return 0;
12 }
```

Lista Ligada

- Outra classificação possível é a lista sem cabeçalho, cujo conteúdo do primeiro elemento tem a mesma importância que os demais. Assim, a lista é considerada vazia se o primeiro elemento for NULL.

Adicionar elementos à lista

- A lista pode ser criada sem nenhum valor, ou seja, vazia.
- Ao inserirmos um elemento na lista ligada, é necessário alocar o espaço na memória, de forma dinâmica, para armazenar o elemento e ligá-lo à lista existente.
- Inserindo um novo elemento no início da lista é a forma mais simples de inserção em uma lista ligada.

Remover elementos da lista

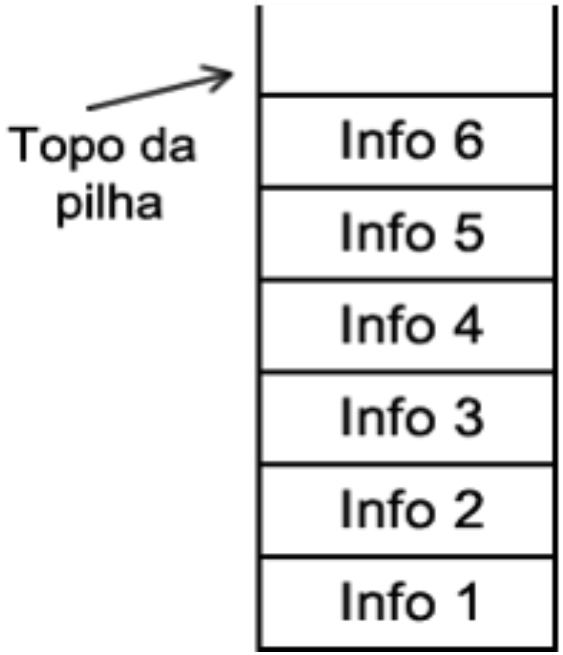
- Com uma lista ligada criada, podemos implementar o uso de funções para a remoção de elementos da lista.
- A função para remover um elemento é mais trabalhosa e complexa, e precisa de informações como parâmetros para remoção, o valor do elemento e a lista. E assim, atualizar o valor da lista sem o elemento removido (CELES, 2004).

Listas Encadeadas

- Quando utilizamos uma lista ligada em vez de um vetor, permitimos que o nosso sistema trabalhe de forma dinâmica com a memória e com a utilização de um ponteiro para indicar qual o próximo elemento da lista.

Definição e elementos de pilhas

- Em programação é fundamental o uso de estruturas de dados para a criação de sistemas, e a mais simples e utilizada é a estrutura de dados do tipo **pilha**.
- Segundo Tenenbaum (2007), uma pilha tem como definição básica um conjunto de elementos ordenados que permite a inserção e a remoção de mais elementos em apenas uma das extremidades da estrutura denominada topo da pilha.



Regras para operação de pilhas

- A Torre de Hanói é um brinquedo pedagógico que representa uma pilha e é muito utilizado em estrutura de dados.
- Tem como objetivo transferir os elementos de uma pilha para outra pilha e sempre mantendo o elemento de maior tamanho abaixo dos elementos de tamanho menor, assim, todos os elementos devem ser movidos de uma pilha a outra, executando a transferência de um elemento por vez, sempre mantendo essa regra.

Regras para operação de pilhas

- Os elementos inseridos em uma pilha possuem uma sequência de inserção, sendo que o primeiro elemento que entra na pilha só pode ser removido por último, após todos os outros elementos serem removidos.

Regras para operação de pilhas

- Segundo Celes (2004), os elementos da pilha só podem ser retirados na ordem inversa da ordem em que foram inseridos na pilha, conhecido como ***LIFO*** (*Last in, first out*, ou seja, o último que entra é o primeiro a sair) ou ***FILO*** (*First in, Last out*, ou seja, o primeiro que entra é o último a sair).

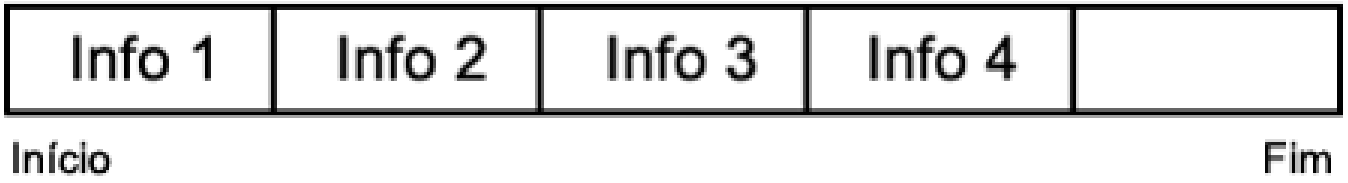
Regras para operação de pilhas

Os passos para criação de uma pilha, segundo esse autor, são:

- Criar uma pilha vazia.
- Inserir um elemento no topo.
- Remover o elemento do topo.
- Verificar se a pilha está vazia.
- Liberar a estrutura de pilha.

Definição e elementos de filas

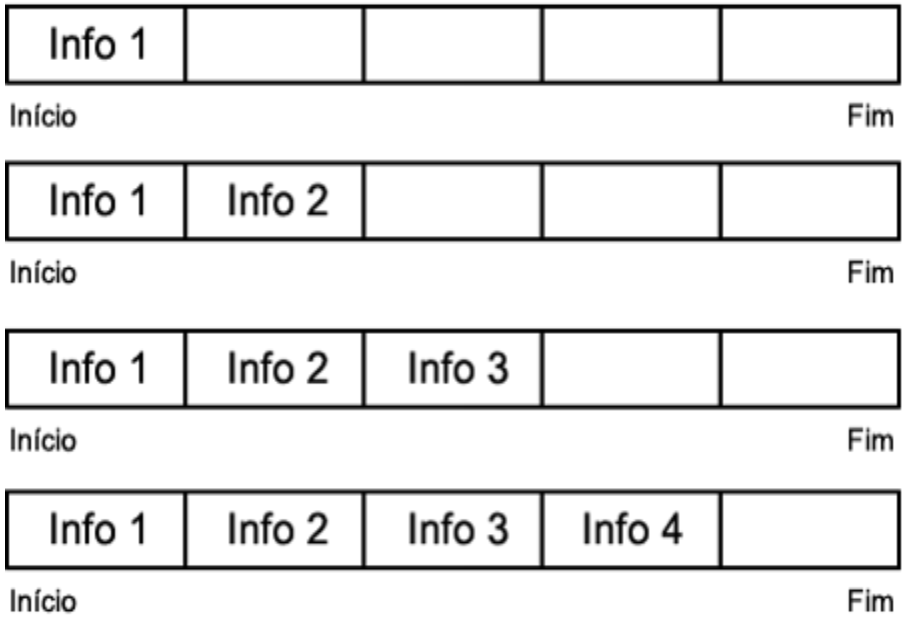
- Outra estrutura de dados muito importante é a estrutura do tipo **fila**. É uma estrutura muito utilizada para representar situações do mundo real.
- Segundo Tenenbaum (2007), uma fila é a representação de um conjunto de elementos no qual podemos remover esses elementos por uma extremidade chamada de início da fila, e pela outra extremidade, onde são inseridos os elementos, é chamada de final da fila.



Regras para operação de filas

- Segundo Silva (2007), em uma fila, os elementos apenas entram por uma extremidade e são removidos pela outra extremidade da fila, conhecido como **FIFO** (*First in, first out*, ou seja, o primeiro que entra é o primeiro a sair).

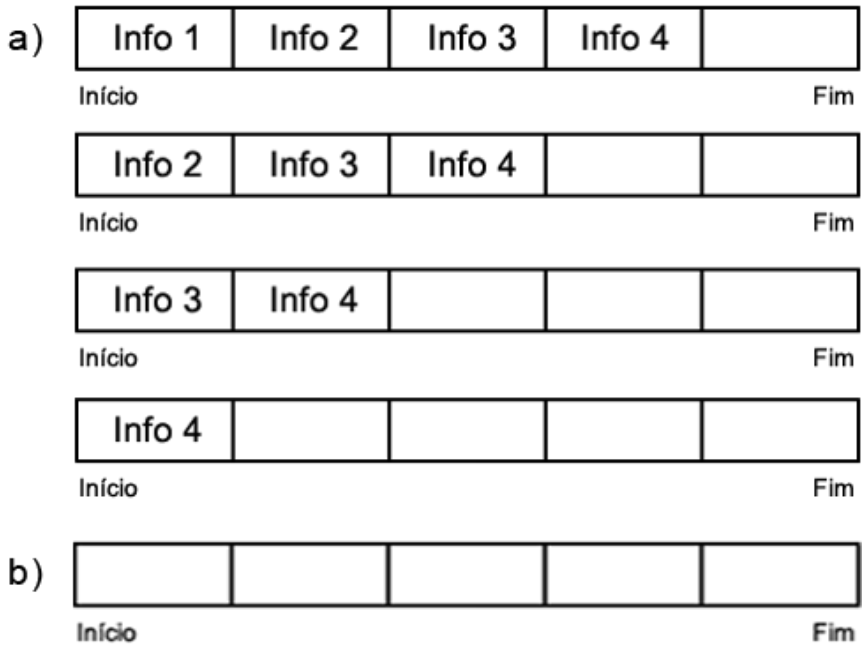
Regras para operação de filas



- No caso desta fila, sabemos quais os elementos com base em seu número de índice, que são as senhas sequenciais. Assim, a fila possui sua ordem de entrada (fim da Fila) e sua ordem de saída dos elementos (início da Fila).

Regras para operação de filas

- A remoção dos elementos dá-se pela outra extremidade contrária (exemplo a).
- Uma fila pode estar no estado de vazia quando não houver nenhum elemento na fila (exemplo b)



Regras para operação de filas

Segundo Celes (2004), os passos para criação de uma fila são:

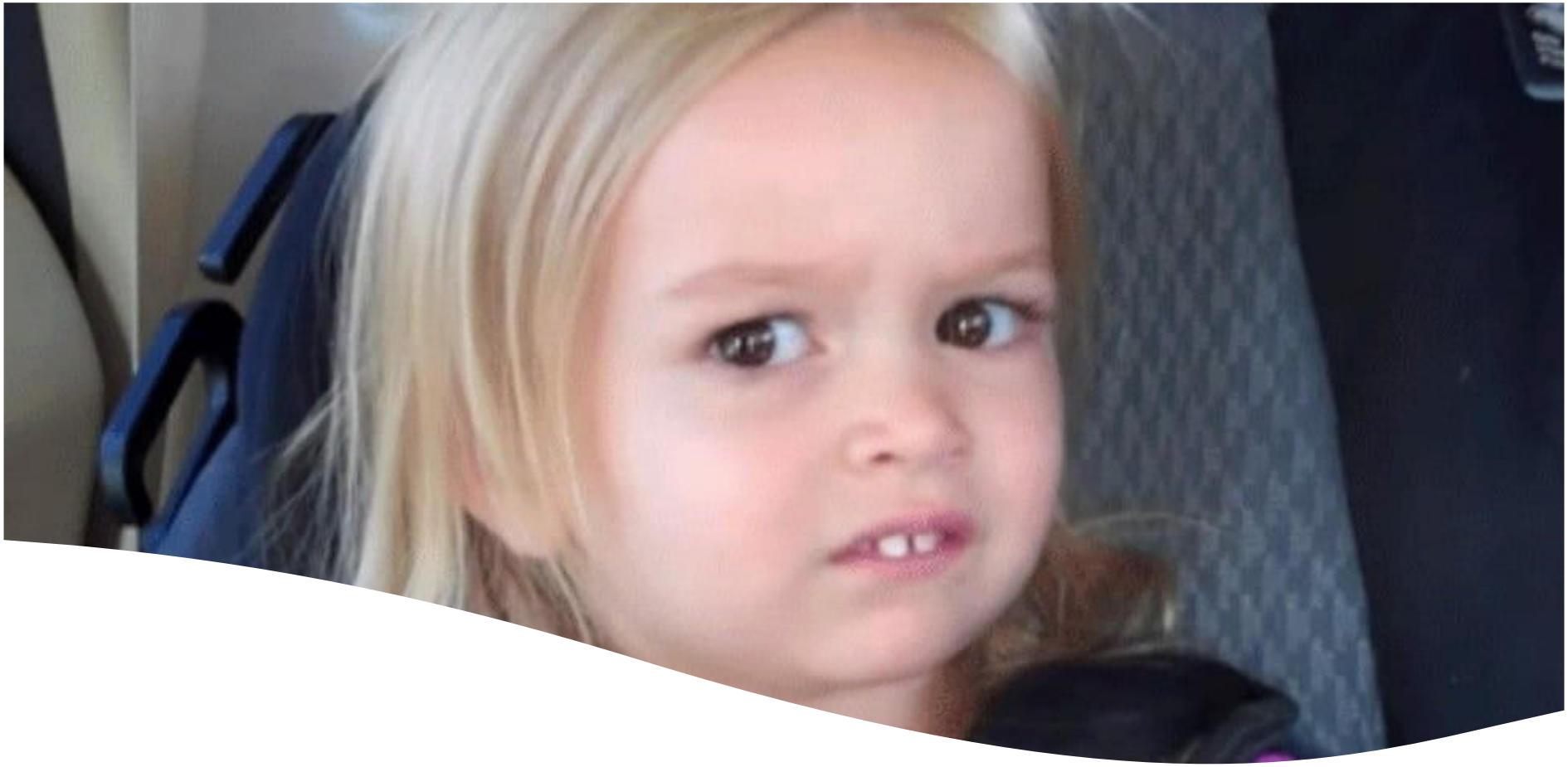
- Criar uma fila vazia.
- Inserir elemento no final.
- Retirar um elemento do início.
- Verificar se a fila está vazia.
- Liberar a fila.

Pilhas x Filas

- As pilhas são muito utilizadas para soluções de problemas onde é necessário o agrupamento de informações, por exemplo, empilhar livros com a mesma letra alfabética.
- No caso de filas, podemos solucionar problemas onde a quantidade de informação é vasta, e assim criar filas para aumentar o fluxo de dados.

Conclusão

- As pilhas estão presentes no nosso dia a dia sem que possamos perceber.
- Tenho certeza de que você já deve ter visto os caminhões do tipo cegonha pelas estradas, por foto ou até mesmo por televisão.
- Os carros que estão no caminhão são inseridos um por vez, e se for preciso remover o primeiro carro colocado no caminhão, será necessário remover o que está atrás dele.



Atividade

Lista de exercícios

uniderp

Lista de exercícios

- Baseado no seu conhecimento adquirido e em pesquisas a serem realizadas desenvolva a Lista02. Prepare seus exercícios no ambiente Dev C++ usando Linguagem C.
- O trabalho pode ser realizado de forma individual ou em duplas.
- Os arquivos deverão ser entregues por e-mail (noiza.waltrick@gmail.com). Assunto: [ED] Lista 02
- Os e-mails enviados após o prazo serão desconsiderados.
- Data de entrega:
  - Turma A: 13/08/2024 até às 23h59.
  - Turma B: 15/08/2024 até às 23h59.

Encerramento

DÚVIDAS, CRÍTICAS E SUGESTÕES, ENVIAR PARA:

noiza@anhanguera.com


uniderp