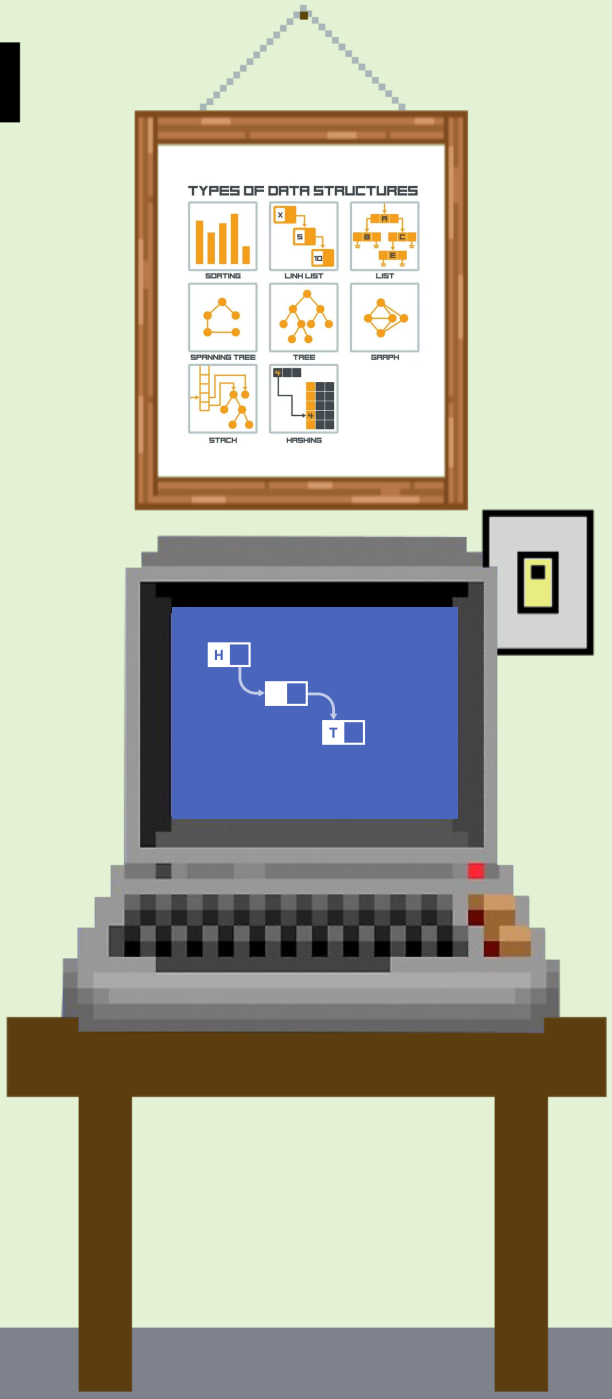
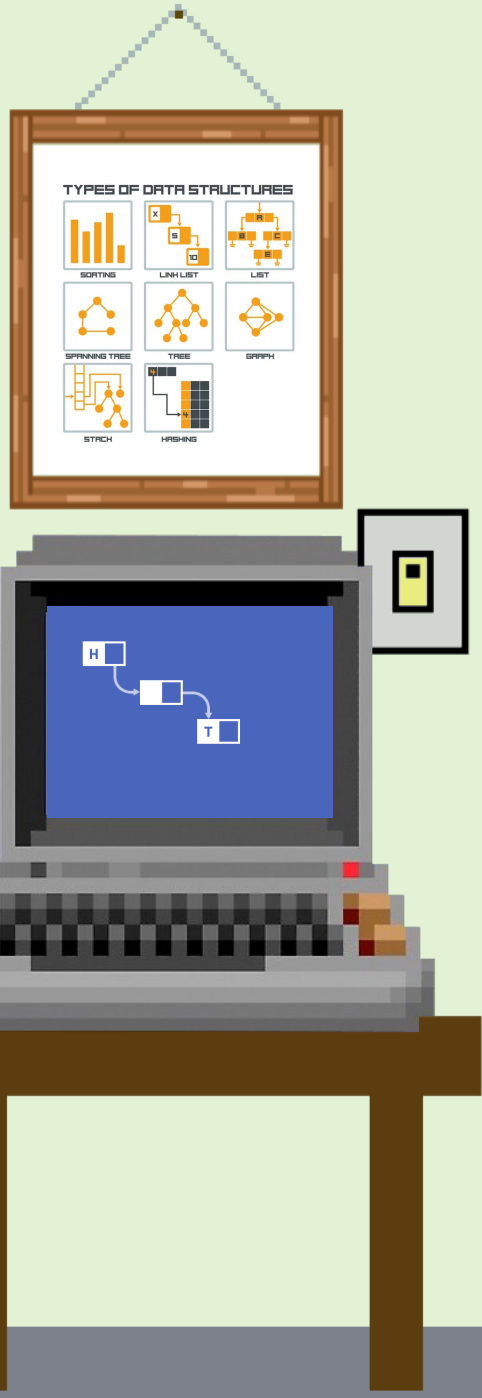
Listas Encadeadas

Estruturas de Dados I



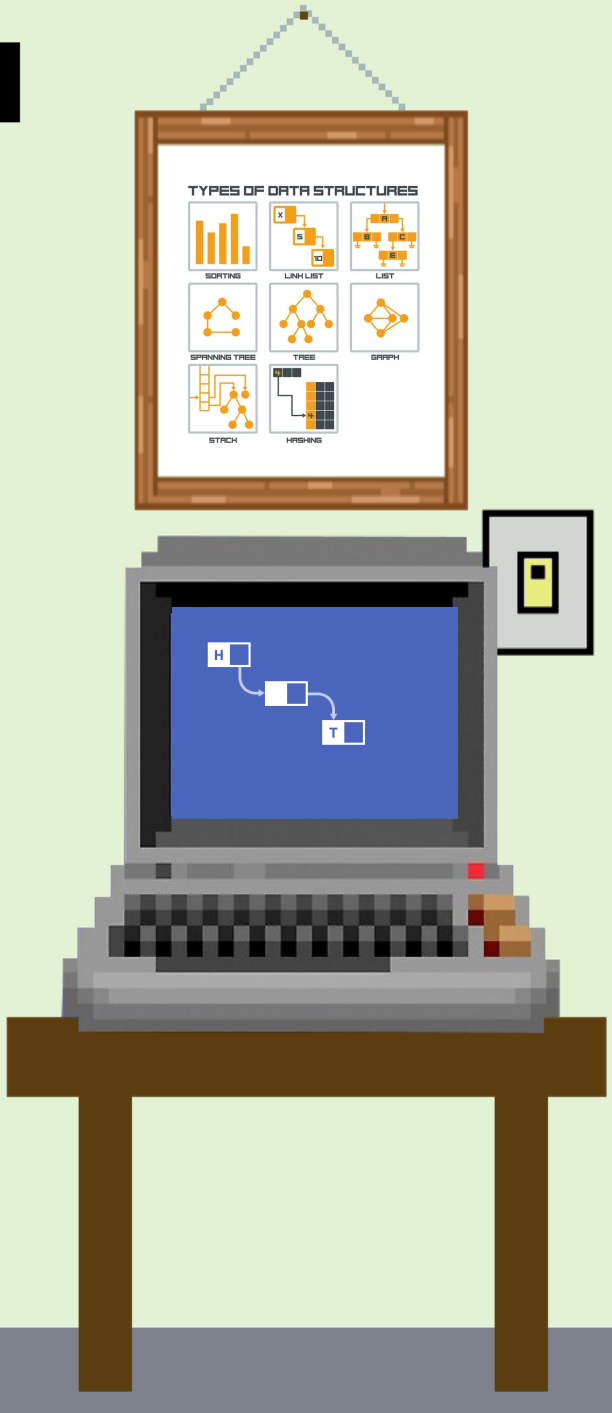
Tipos Abstratos de Dados

- Especifica o tipo de dado (domínio e operações) sem referência a detalhes da implementação
- Minimiza código do programa
  - Mais liberdade para mudar a implementação com menor impacto nos programas
- Os programas que utilizam TAD não "conhecem" a forma como as estruturas são implementadas
- Listas
- Pilhas
- Filas
- Árvores



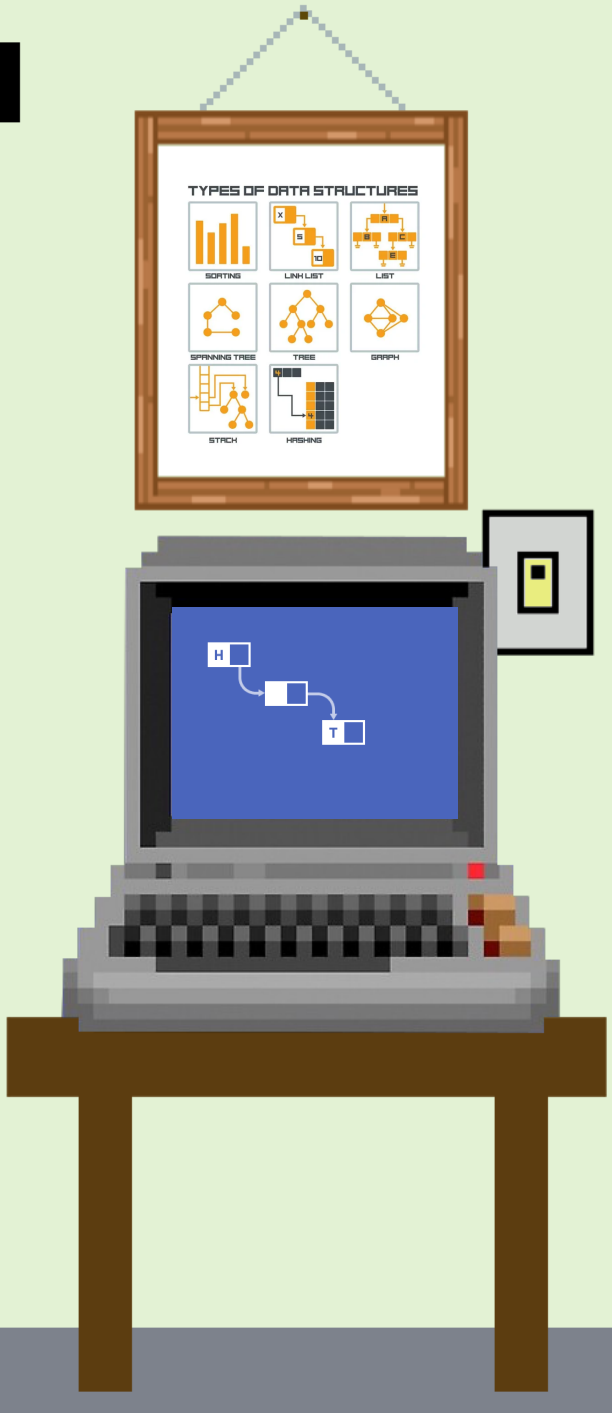
Listas Encadeadas

- Interliga os elementos de um conjunto
- Podem crescer ou diminuir de tamanho durante a execução de um programa (conforme a demanda)
- Podem ser concatenadas ou divididas
- Listas ligadas
  - Simples
  - Duplas
  - Circulares



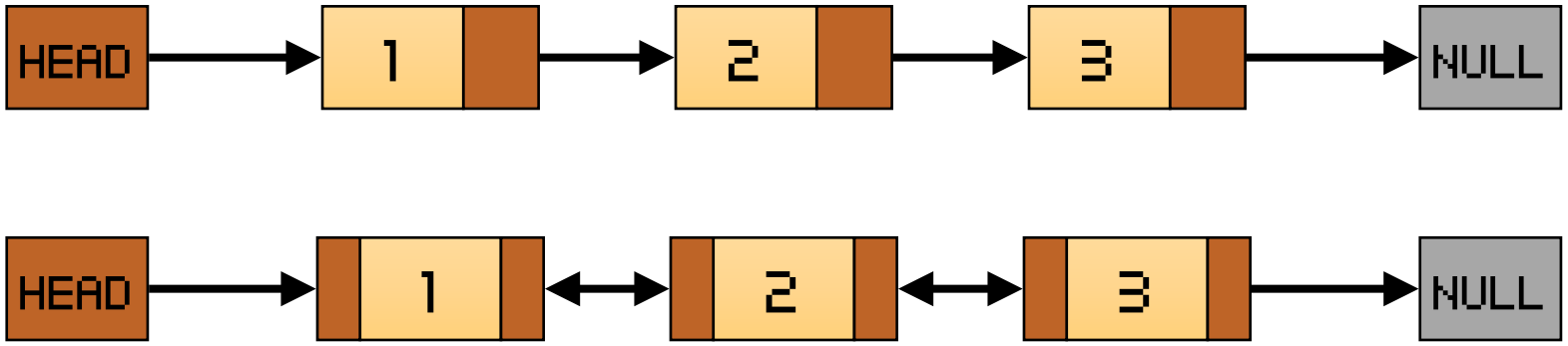
Listas Encadeadas Simples

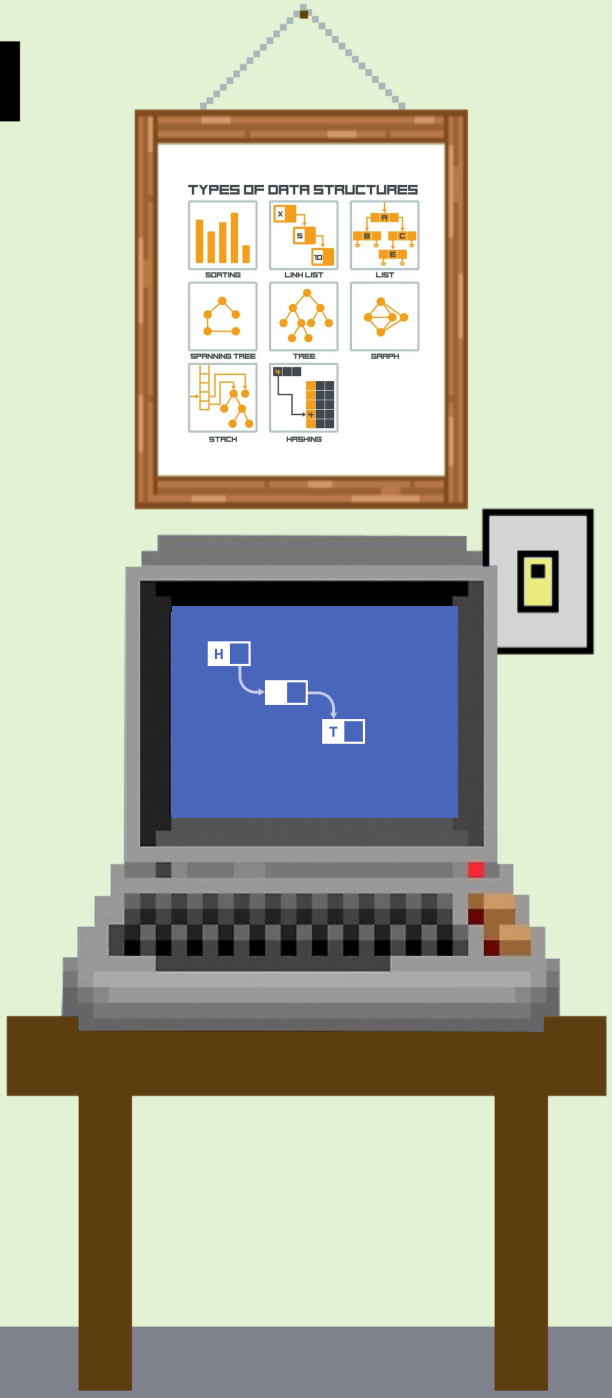
- Vantagens
  - Custo constante para inserir ou retirar itens do meio da lista (Importante para listas ordenadas)
  - Ideais quando não se sabe a demanda por memória
- Desvantagens
  - Requer memória extra para armazenar os ponteiros
  - Gerenciamento de memória (Alocação, Liberação)
  - O acesso de itens pela posição requer busca



Listas Encadeadas

- Constituída por células
- Representação de uma sequência de objetos
  - Todos os elementos são do mesmo tipo
  - Cada célula contém o endereço da célula seguinte
  - Células também podem conter o endereço do elemento anterior (listas duplamente encadeadas)

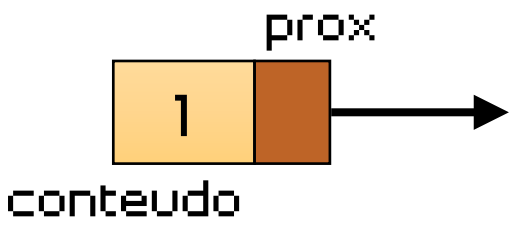


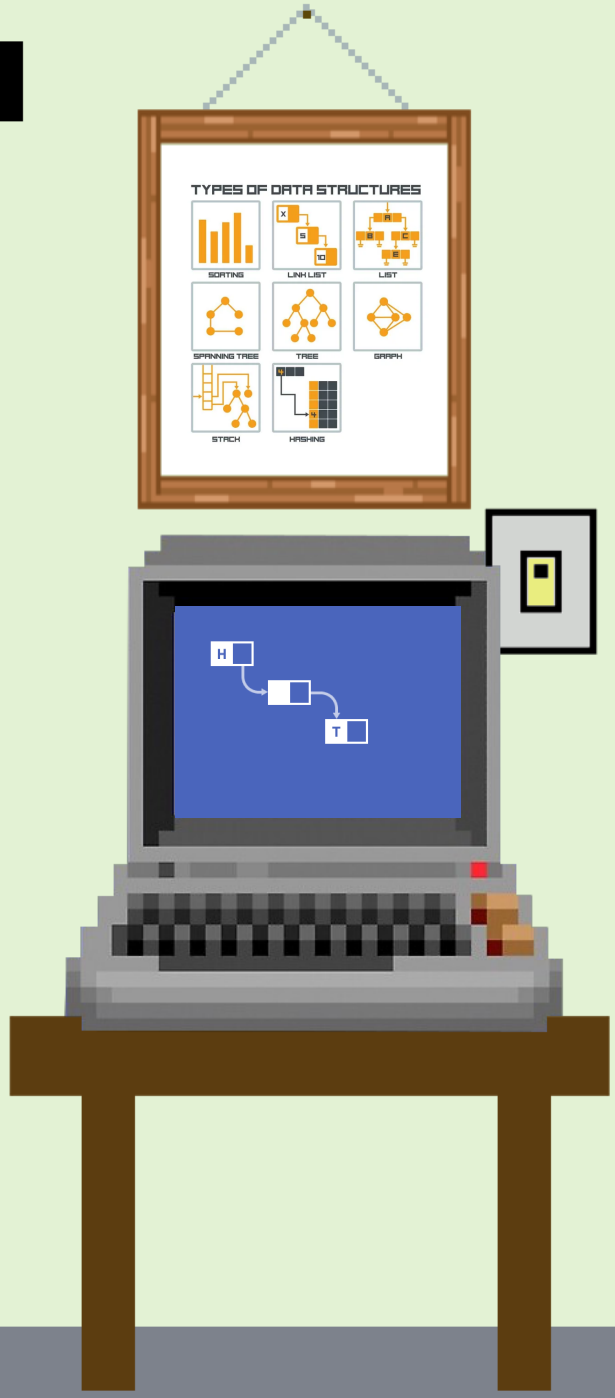


Listas Encadeadas Simples

- Células armazenam apenas seu conteúdo e o endereço da célula seguinte
- A exemplo dos vetores, acessamos as listas a partir do endereço do primeiro elemento (head ou cabeça)
- A última célula da lista deve apontar para NULL
- Tratamos as células como um novo tipo de dados

```
typedef struct cel {
        int conteudo;
        struct cel *prox;
} celula;
```





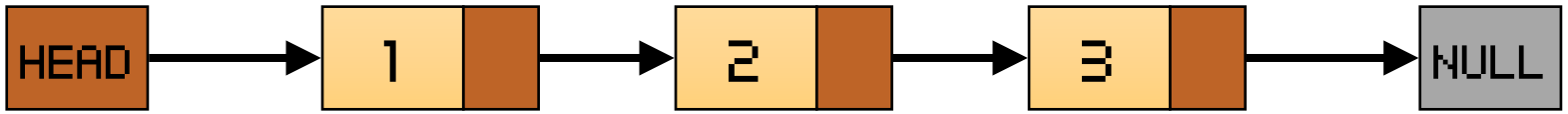
Listas Encadeadas Simples

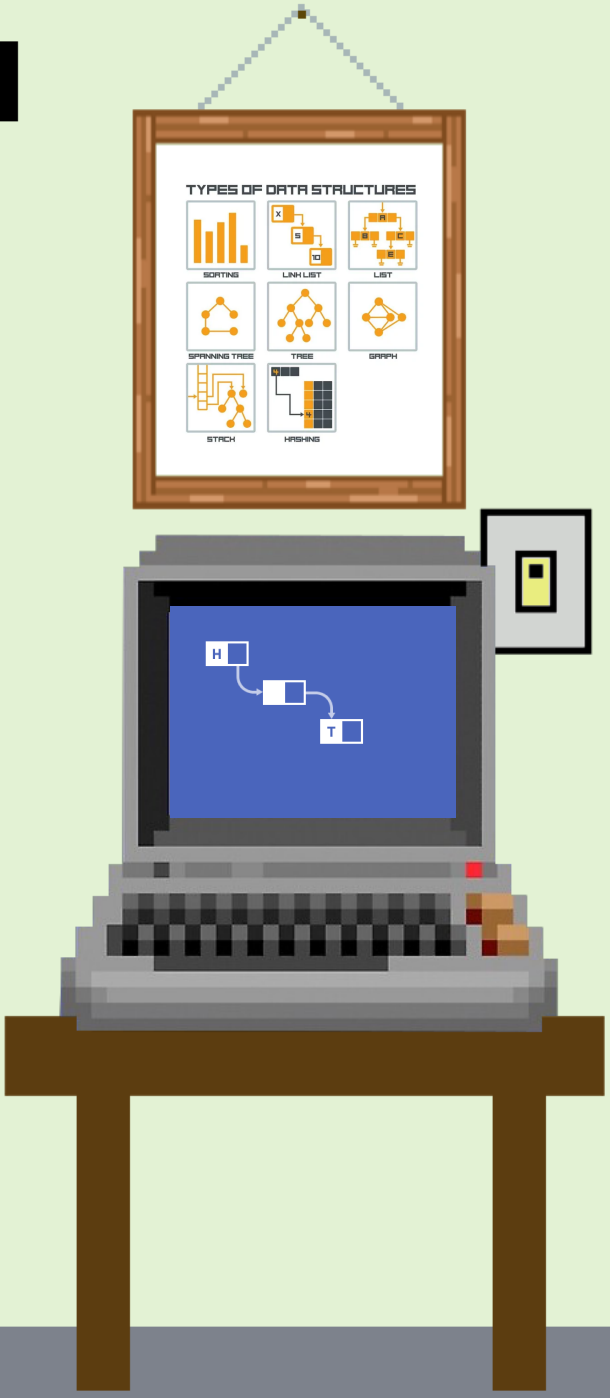
- Acesso aos campos

  celula c;

  celula *p;

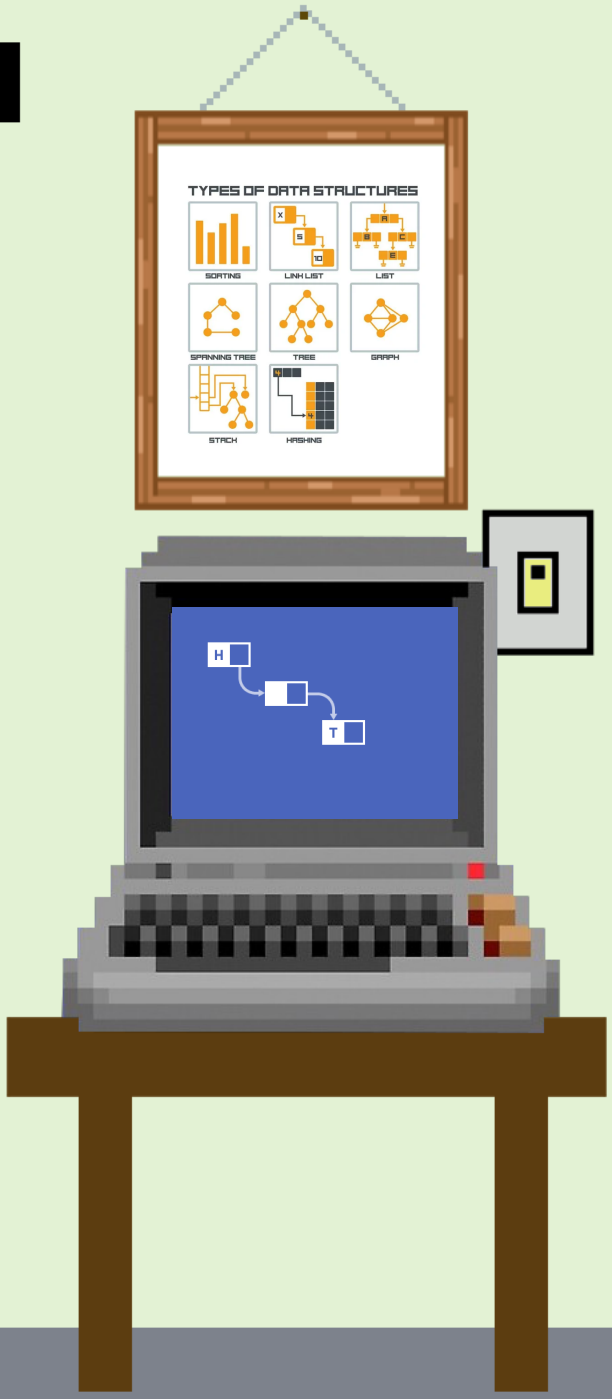
  - Conteúdo
    - c.conteudo //conteúdo da célula
    - p->conteudo //conteúdo da célula
  - Endereço
    - c.prox //endereço da próxima célula
    - p->prox //endereço da próxima célula





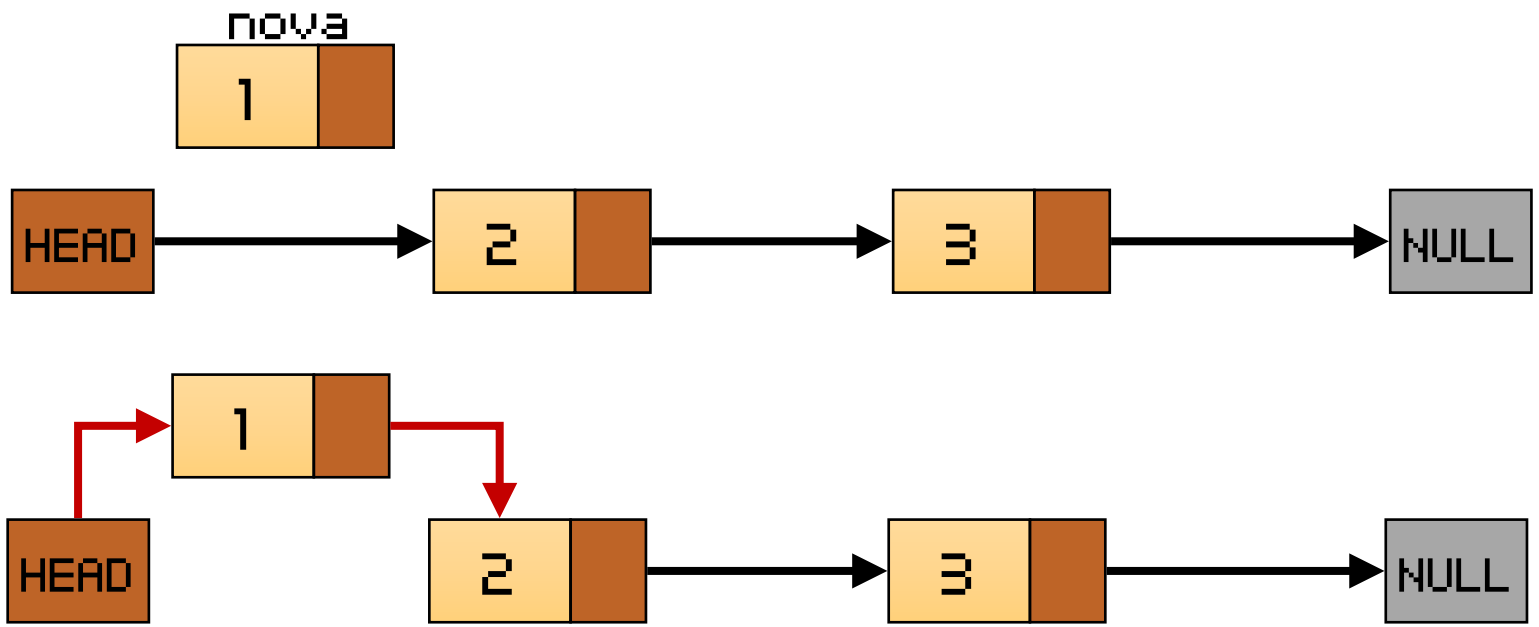
Principais Operações

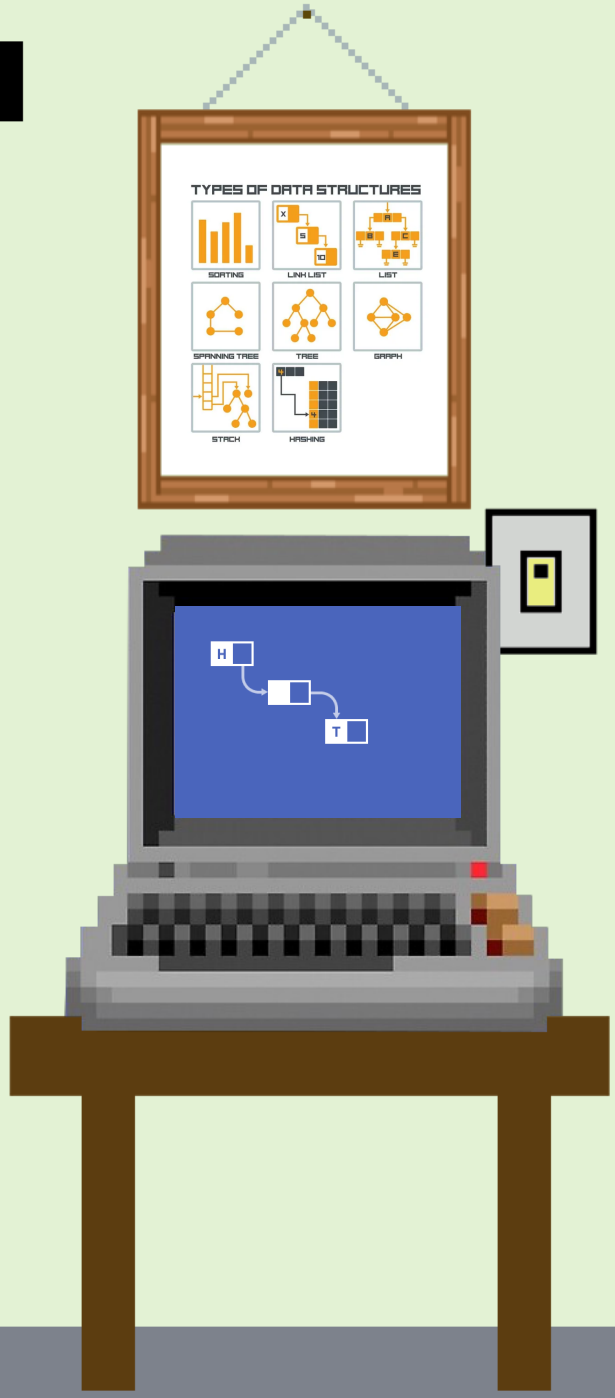
- Inserir elemento
  - Início
  - Meio
  - Fim
- Imprimir lista
- Buscar elemento
- Editar elemento
- Remover elemento



Inserir

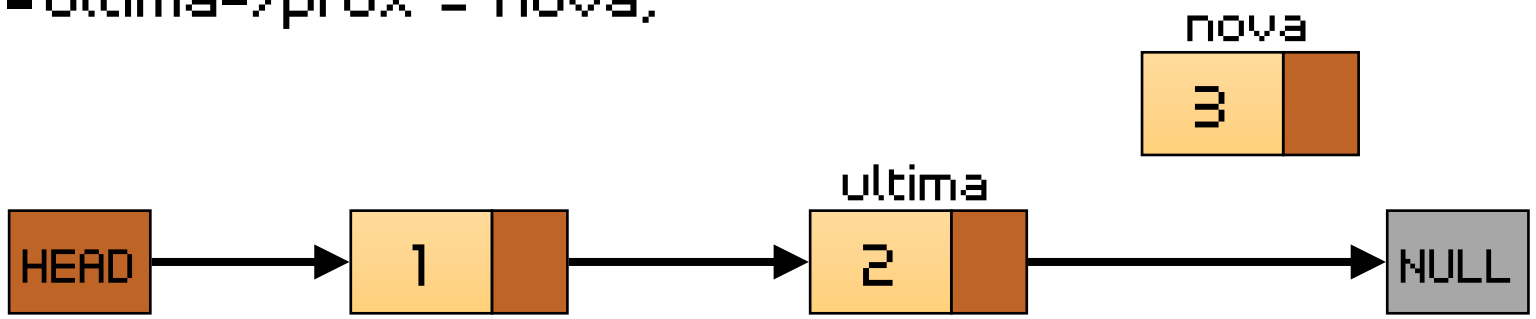
- Inicio
  - nova->prox = *ptr_cabeca;
  - *ptr_cabeca = nova;

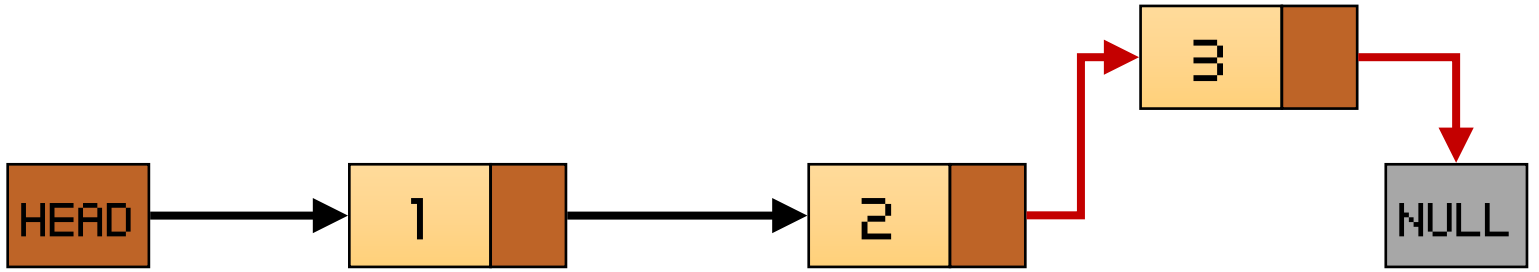


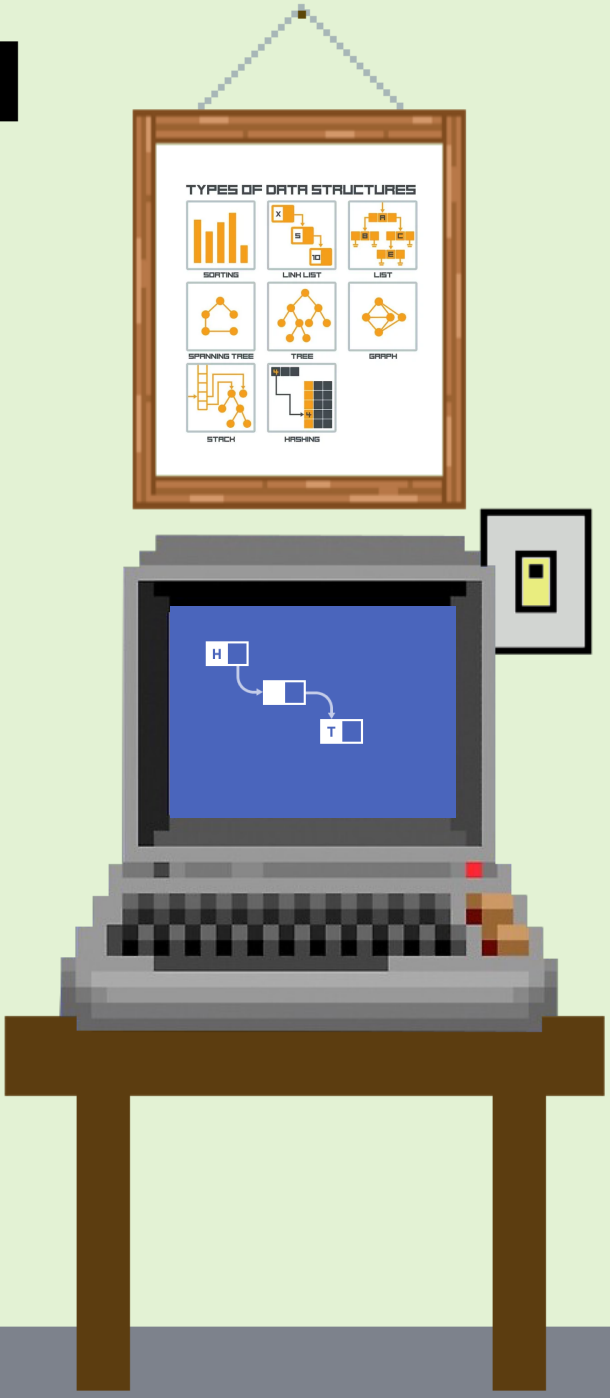


Inserir

- Fim
  - nova->prox = NULL;
  - ultima->prox = nova;

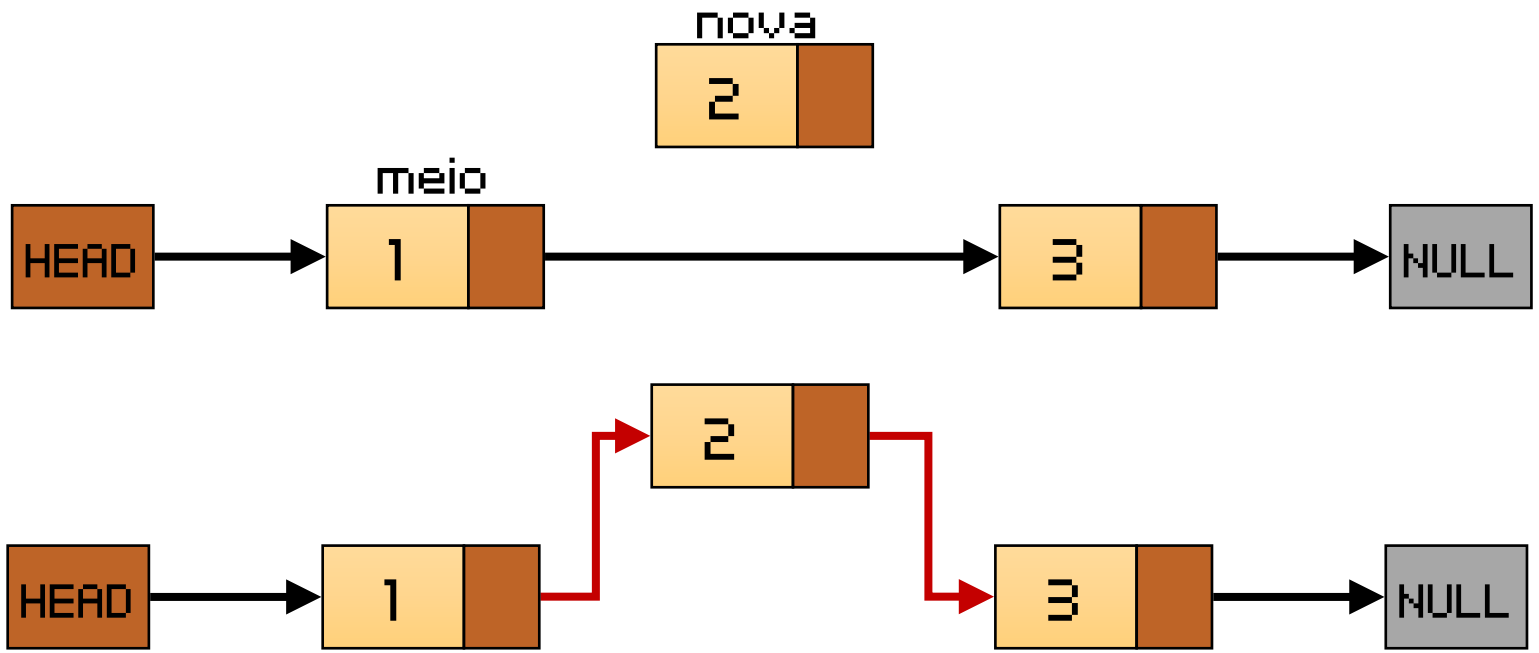


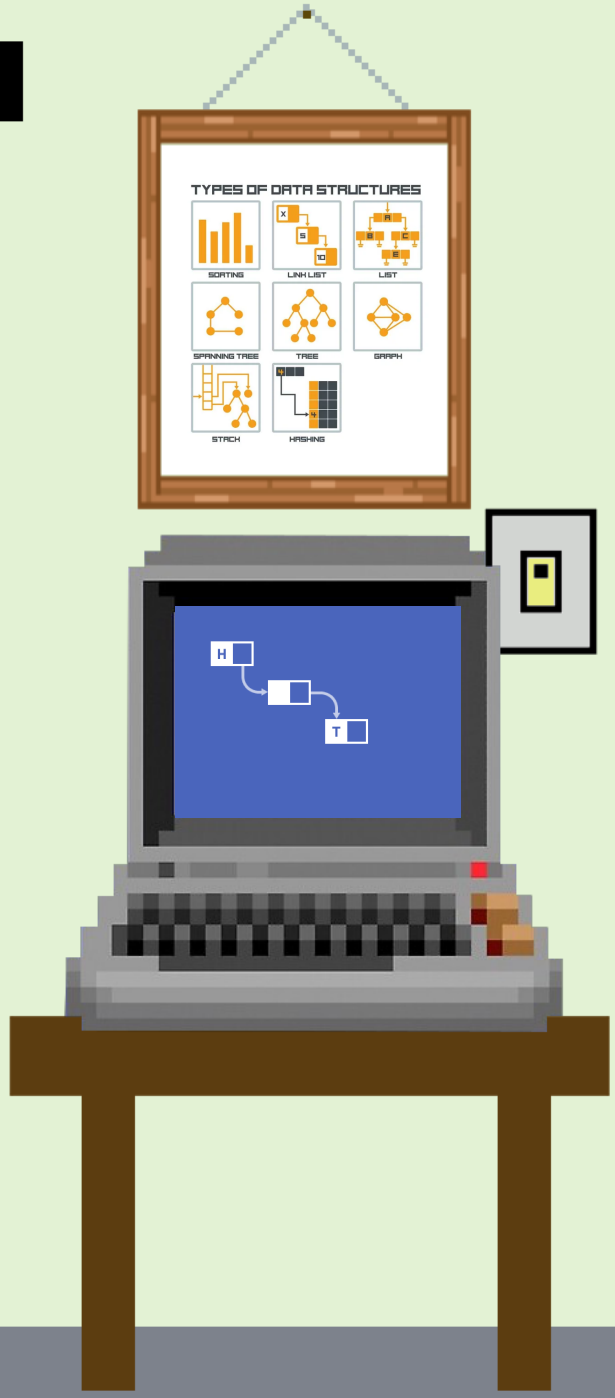




Inserir

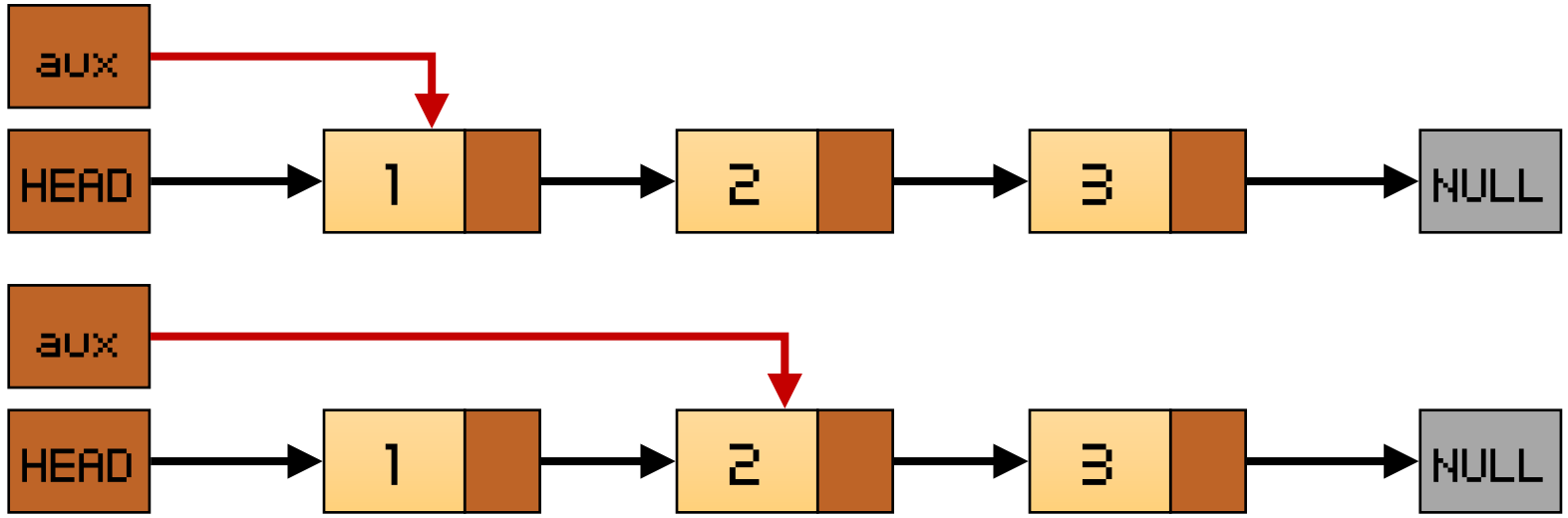
- Meio
  - nova->prox = meio->prox;
  - meio->prox = nova;

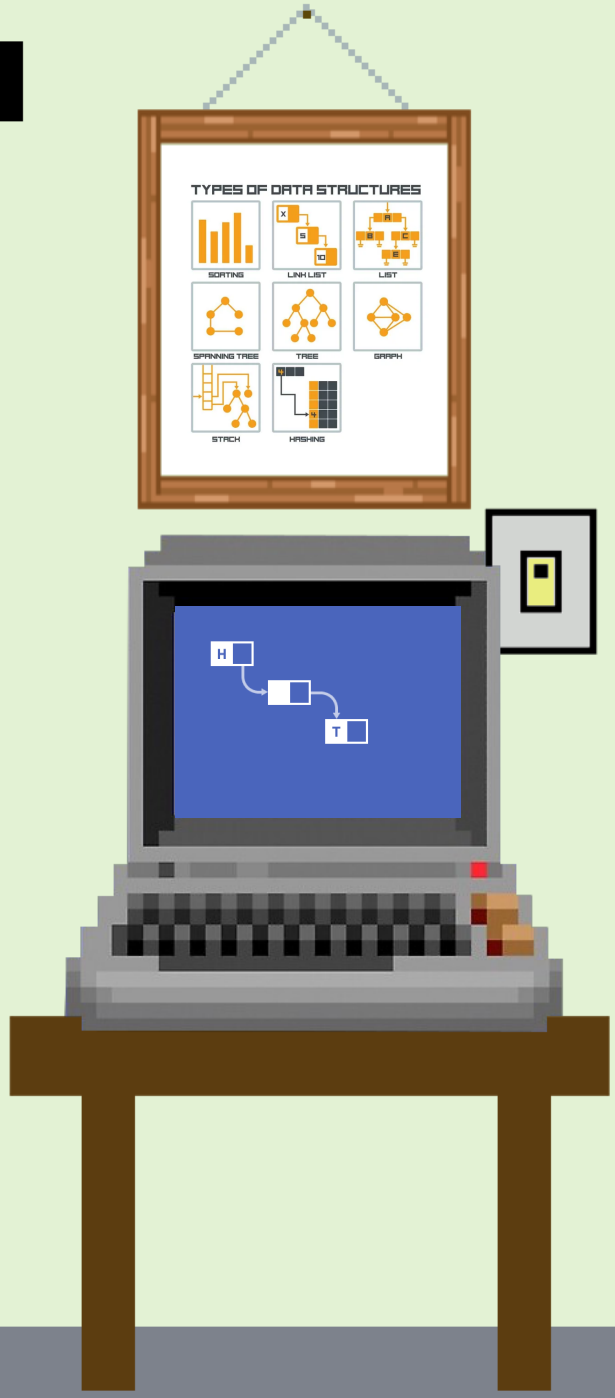




Imprimir

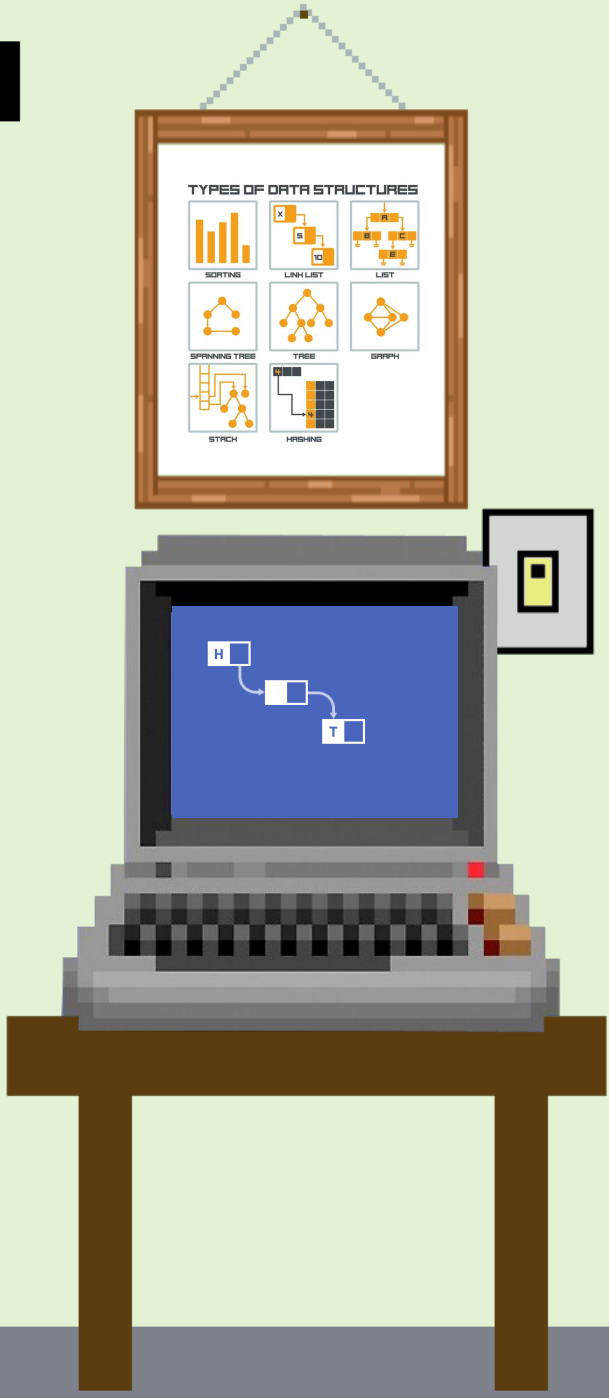
```
celula *aux = cabeca;
while (aux != NULL) {
    printf("Conteudo: %d\n", aux->conteudo);
    aux = aux->prox;
}
```





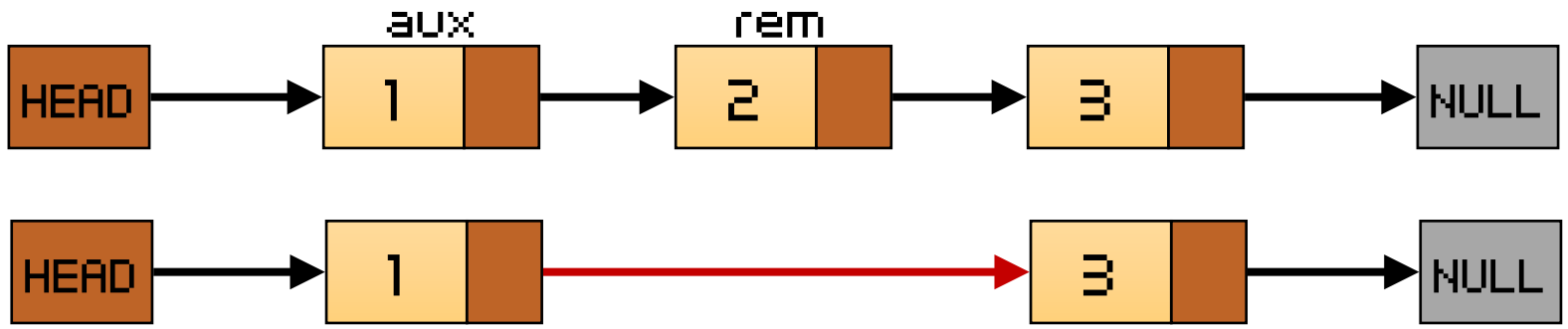
Buscar

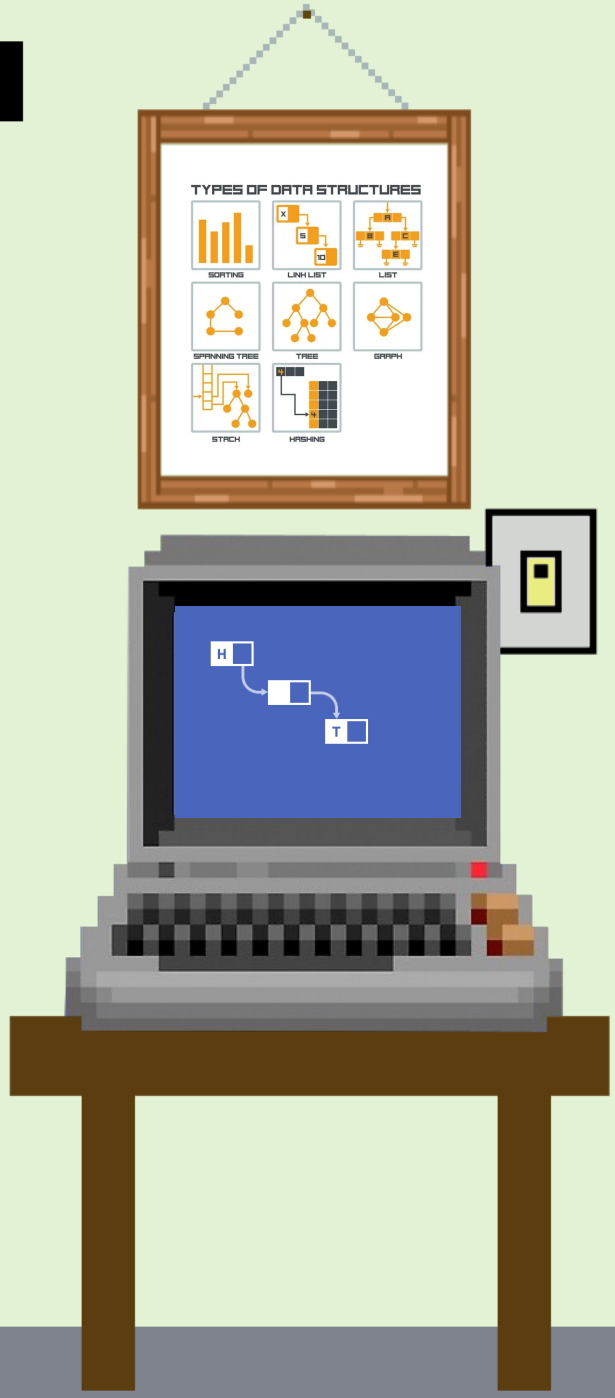
```
celula *aux = cabeca;
while (aux != NULL) {
    if (aux->conteudo == valor) {
      return aux;  //"Elemento encontrado"
    }
    aux = aux->prox;
}
return NULL;  //"Elemento não encontrado"
```



Remover

```
celula *aux = cabeca;
while (aux != NULL) {
    if (aux->prox == rem) {
      aux->prox = rem->prox;
      free(rem);
    }
    aux = aux->prox;
}
```





Atividade

- Implemente as funções descritas abaixo sobre listas encadeadas simples
  - Retornar a posição de um elemento na lista
  - Localizar um elemento da lista e trocar de posição com o próximo
  - Receber duas listas e concatenar em uma só
  - Receber uma lista e dividir em duas na metade
  - Reverter uma lista (primeiros ficam por último)