Parte 1 – Teoria

1. O que significa alocação estática de memória para um conjunto de elementos?
A alocação estática de memória para um conjunto de elementos significa que o espaço de memória necessário para armazenar esse conjunto é determinado em tempo de compilação ou no início da execução do programa e permanece fixo durante toda a sua vida útil. O tamanho não pode ser alterado dinamicamente.
Referência, Universidade Federal do Paraná (UFPR). Alocação de memória: estática e automática.

2. Qual a diferença entre alocação estática e alocação dinâmica?
A diferença essencial reside no **controle sobre o tamanho da memória em tempo de execução**:

- **Alocação Estática:** O tamanho da memória é **predefinido** e **fixo** pelo compilador. Uma vez alocada (geralmente na *Stack*), não pode ser aumentada ou diminuída, o que pode levar ao desperdício de recursos.
- **Alocação Dinâmica:** O espaço pode ser alocado e **redimensionado** (*aumentar ou diminuir*) **durante a execução do programa** (*runtime*), utilizando apenas a quantidade necessária de memória no *Heap*.

Referência, Linguagem C. Alocação Dinâmica em C.

3. O que é um ponteiro?
Um **ponteiro** é uma variável cujo valor armazena o **endereço de memória** de outra variável ou de uma estrutura de dados. Ele permite o acesso indireto aos dados e é fundamental para a alocação dinâmica e estruturas encadeadas.
Referência, Wikipédia. Ponteiro (programação).

4. O que é uma estrutura de dados homogêneos?
Uma **estrutura de dados homogêneos** é aquela onde **todos os elementos armazenados são do mesmo tipo de dado** ou natureza. Exemplos incluem vetores e matrizes.
Referência, Amanda Nascimento (Artigo Técnico). Dados homogêneos X heterogêneos.

5. O que é uma estrutura de dados heterogêneos?
Uma **estrutura de dados heterogêneos** é aquela que armazena elementos de **tipos de dados diferentes** em uma única unidade estrutural. O exemplo clássico são os *Structs* (registros).

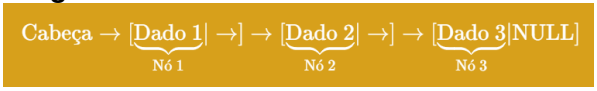
Referência, Amanda Nascimento (Artigo Técnico). Dados homogêneos X heterogêneos.

6. Qual a vantagem das listas sobre os vetores em termos de consumo de memória? Exemplifique.
A principal vantagem é a **eficiência de uso da memória** devido à natureza dinâmica das listas. Listas encadeadas **alocam memória sob demanda** (apenas para os nós existentes), evitando o **desperdício** que ocorre em vetores estáticos, que precisam ser dimensionados para o tamanho máximo, mesmo que a ocupação real seja muito menor.
Exemplo: Um vetor reservado para 1.000 posições que usa apenas 10 desperdiça o espaço das 990 posições; uma lista aloca apenas para os 10 nós e seus ponteiros.

Referência, Universidade de São Paulo (IME-USP). O que é uma lista encadeada e como implementá-la.

7. O que é uma lista simplesmente encadeada? Apresente um diagrama para ilustrar essa estrutura de dados.
Uma **lista simplesmente encadeada** é uma coleção linear de nós onde cada nó contém o **dado** e um único **ponteiro** que aponta para o **próximo** nó da sequência. O percurso só é possível em uma direção.

Diagrama:



Referência, Universidade de São Paulo (IME-USP). O que é uma lista encadeada e como implementá-la.

8. O que é uma lista duplamente encadeada? Apresente um diagrama para ilustrar essa estrutura de dados.

Uma **lista duplamente encadeada** é uma coleção linear de nós onde cada nó armazena o **dado** e **dois ponteiros**: um para o **próximo** nó e um para o **nó anterior**. Isso permite o percurso em ambas as direções.

Diagrama:

Referência, Universidade Federal do Paraná (UFPR). Programação: Listas Duplamente Encadeadas.

9. O que é uma lista duplamente encadeada? Apresente um diagrama para ilustrar essa estrutura de dados.

(Pergunta idêntica à 8). Uma **lista duplamente encadeada** é uma coleção linear de nós onde cada nó contém ponteiros para o nó anterior e para o próximo, permitindo o percurso bidirecional.

Referência, Universidade Federal do Paraná (UFPR). Programação: Listas Duplamente Encadeadas.

10. Explique o funcionamento do algoritmo de busca binária e sequencial.

- **Busca Sequencial:** Percorre a lista **elemento por elemento**, comparando cada um com o valor procurado, do início ao fim. Funciona em listas ordenadas ou não, com complexidade O(n).
- **Busca Binária:** Compara o valor procurado com o elemento **central** da lista. A cada comparação, descarta metade da lista (inferior ou superior) e repete o processo na metade restante. **Requer que a lista esteja ordenada**, com complexidade O(log n).

Referência, Universidade de São Paulo (IME-USP). Busca sequencial e binária.

11. Explique o funcionamento dos seguintes algoritmos de ordenação: Insertion sort, Selection sort, Merge sort, Count sort, Quicksort.

- **Insertion Sort:** Pega um elemento da sublista não ordenada e o **insere na posição correta** da sublista já ordenada. Complexidade: $O(n^2)$.
- **Selection Sort:** A cada iteração, **seleciona o menor elemento** da sublista não ordenada e o **troca** para a primeira posição dessa sublista. Complexidade: $O(n^2)$.
- **Merge Sort:** Algoritmo de Divisão e Conquista. **Divide** a lista em metades recursivamente e, em seguida, **intercala** (*merge*) as sublists ordenadas. Complexidade: $O(n \log n)$.
- **Count Sort:** Algoritmo não comparativo. **Conta a frequência** de cada valor (em um intervalo k) e usa essa contagem para posicionar os elementos diretamente no vetor de saída. Complexidade: O(n+k).
- **Quicksort:** Algoritmo de Divisão e Conquista. Escolhe um **pivô** e **particiona** a lista em elementos menores e maiores que o pivô. Ordena as partições recursivamente. Complexidade: $O(n \log n)$ (caso médio).

Referência, FreeCodeCamp. Algoritmos de ordenação explicados com exemplos...