ESTRUTURA DE DADOS

Conceitos Básicos:

Dado

Um dado pode ser definido como um valor bruto e sem significado. 10, "b", 2.6, true são exemplos de dados.

Dados são a base da computação, mais precisamente a manipulação deles. Tudo que um computador faz é manipular dados para nos auxiliar na tomada de decisões. Podemos dizer que dados podem ser originados a partir da interação entre usuários humanos com computadores (softwares) ou da interação direta entre computadores (softwares), seja apenas através de trocas ou da criação de novos dados a partir da manipulação de dados preexistentes.

Estrutura

Uma estrutura pode ser definida como a forma como um conjunto de dados pode ser armazenado e manipulado.
A depender do tipo de dado, cada estrutura é armazenada e manipulada de uma maneira específica.
Assim, podemos dizer que cada estrutura define uma álgebra, um conjunto de operações/manipulações que são permitidas sobre esta estrutura.

Estrutura de Dados

Um conjunto de teorias e práticas responsáveis por definir a forma como os dados podem ser armazenados, representados e consequentemente manipulados.

Informação

Pode-se dizer que ela surge quando um dado ganha um significado, uma semântica a partir ou para um determinado contexto.

Tipos de Dados e Tipos Abstratos de Dados

Tipo de dado (TD) pode ser definido como as categorias nas quais os dados podem se enquadrar:

1) Tipos Primitivos de Dados:

Os TD estão ligados às linguagens de programação, e elas podem disponibilizar maiores subdivisões ou TDs próprios.
Algumas linguagens possuem, ainda, uma "tipagem forte" ou "tipagem fraca". Isso diz respeito ao momento em que devemos informar o tipo de dado que a variável deve ser.

São divididos:

Numéricos - inteiros e reais
Lógicos - true e false
Literais - textos ou caracteres

2) Tipos Abstratos de Dados

O tipo abstrato de dado (TAD) é um tipo de dado construído a partir de tipos primitivos (TDs) e tem como finalidade representar estruturas do mundo concreto (real) para o mundo abstrato (computacional).
Assim, podemos afirmar que TADs — ao contrário dos TDs — não são ligados às linguagens de programação, embora possam ser representados por elas.
Como dito, TADs são estruturas (conceitos/entidades) que pertencem a um contexto que deve ser modelado para poder ser manipulado pelo computador.

O termo "abstrato" é utilizado no sentido de ausência de detalhes, evitando uma descrição minuciosa do mundo concreto.
Mesmo com essa "ausência de detalhes", os TADs são capazes de alcançar os objetivos almejados no mundo computacional.
Essa simplificação não diminui a capacidade de representação, apenas torna o processo de representação mais fácil de entender pelo homem e pelos computadores. Essa simplificação é vital para tornar o mundo real "computável".

Podemos dizer que TAD pode ser considerado um TD, no caso um tipo Complexo, que pode ser encarado como uma evolução dos tipos primitivos, pois é constituído a partir destes e possui um poder representacional maior.

São divididos em:

Lineares
Hierárquicos
Mapas
Conjuntos

Ponteiros

É uma forma de acessar o dados através do local na memória do computador onde eles se encontram. Ponteiros possibilitam acesso ao valor do dado e também ao seu endereço de memória.
Ponteiros são a base para a criação das estruturas de dados. Com o uso deles, é possível alocar e desalocar memória dinamicamente.

Sempre estamos usando ponteiros, seja de forma direta ou indireta. Linguagens mais modernas (como Java, C# e Python) não possibilitam o uso direto de ponteiros; entretanto, toda vez que utilizamos uma variável que é do tipo de algum objeto (seja criado pelo programador ou disponibilizado pela própria linguagem), estamos usando um ponteiro indiretamente. Ou seja, por "debaixo dos panos", tais linguagens usam ponteiros para manipular objetos. Elas só fornecem um nível de abstração maior, que nos poupa de manusear diretamente ponteiros, eliminando possíveis erros decorrentes de acessos indevidos a determinados locais na memória.

Memória

Pode-se dizer que ela é "*um componente capaz de armazenar dados e programas (softwares)*". É nesse componente que os softwares são carregados para execução, assim como os dados que são manipulados pelos softwares.

Tipos:

1) Memória Principal (MP)

Acesso rápido e curto tempo de vida (volátil) = RAM

É nesse tipo de memória que o *heap* e a *stack* são alocados, pois programas entram e saem de execução constantemente.
Esse processo de alocação da memória para ser usada pelo *heap* ou *stack* pode ser feito de forma estática ou dinâmica, a depender da necessidade.

Na alocação estática, a memória de que um *tipo de dado* ou programa possa vir a necessitar é alocada **toda de uma vez e de forma sequencial**, sem considerar que toda ela **não seria realmente necessária** na execução do programa.

A alocação dinâmica, por sua vez, aloca a memória **sob demanda** e de forma não sequencial. Assim, os **espaços de memória** podem ser **alocados, liberados ou realocados para diferentes propósitos durante a execução do programa**

2) Memória Secundária (MS)

Acesso mais lento e longo tempo de vida = ROM (Memória de leitura apenas)
Ex. firmware de um dispositivo, as instruções de boot do sistema operacional.

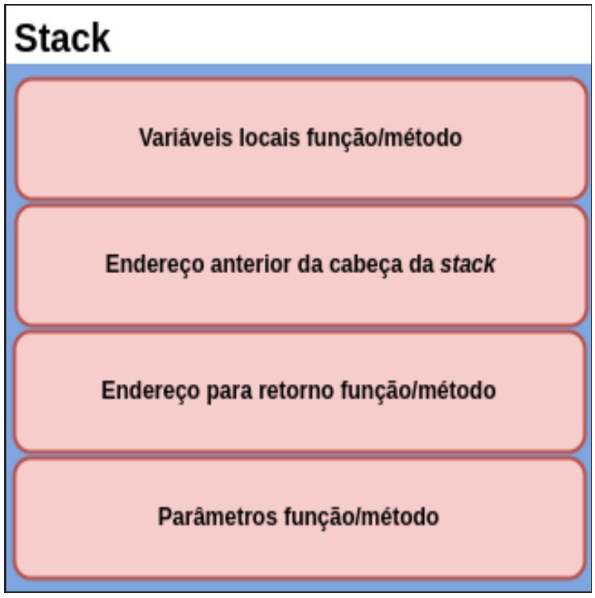
Heap

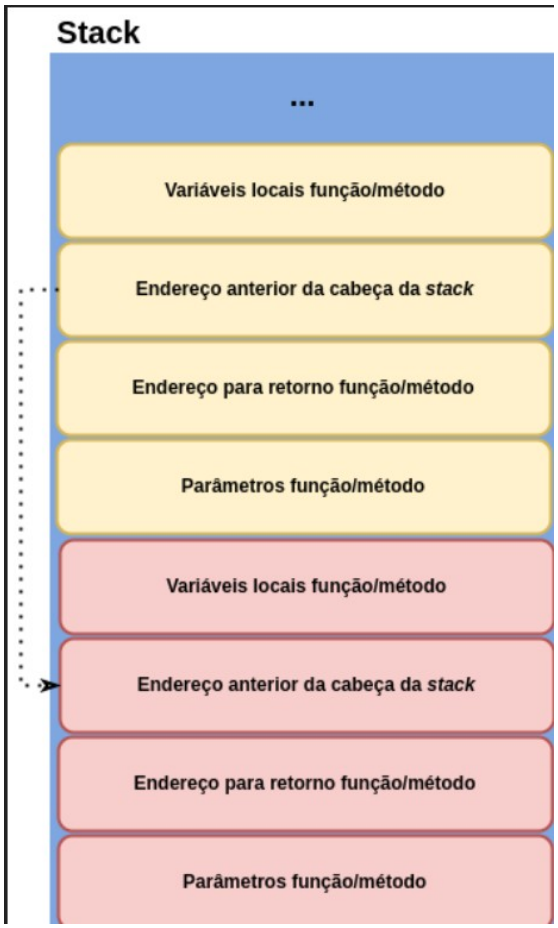
O *heap* é o principal espaço de memória utilizado pelos computadores para executar programas.
É nesse local que linguagens estruturadas armazenam as **variáveis de escopo global** e que linguagens orientadas a objetos **armazenam os objetos** criados através do operador *new*. É também nesse l**ocal que o programa é carregado para poder ser executado**, assim como todo o **espaço inicial de memória que precisar ser alocado**, inclusive *stacks*.
Esse espaço de memória utiliza constantemente a alocação dinâmica sob demanda para possibilitar a execução de programas.

Stack

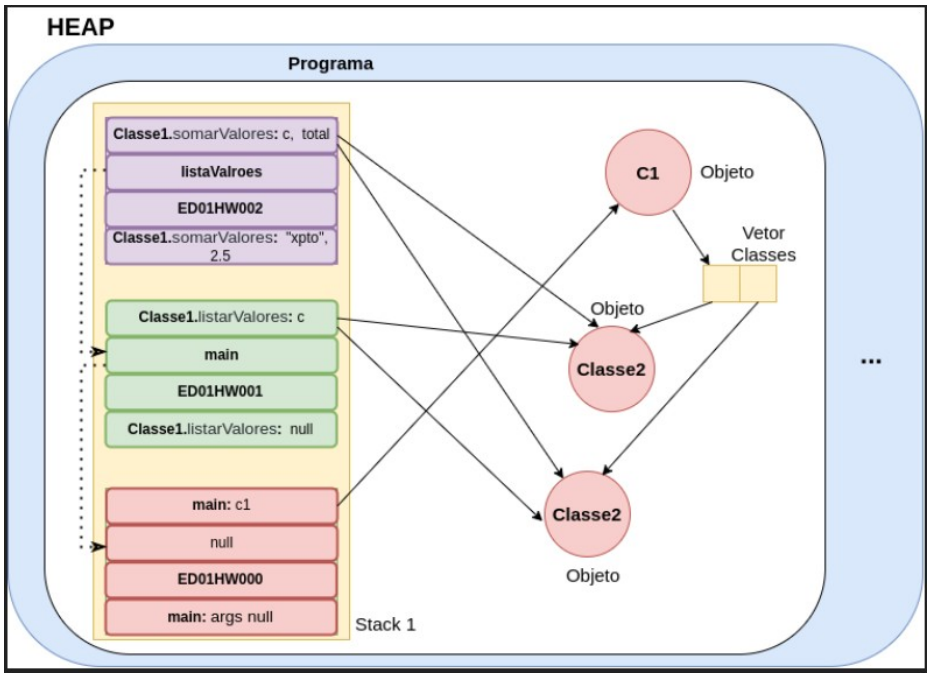
A *stack*, como dito no parágrafo anterior, faz parte do *heap*. Ela é criada — ou são criadas — no momento de carregamento do programa, e é usada para possibilitar que mudanças de contexto sejam feitas durante a execução do programa e a linha principal de execução consiga seguir seu fluxo normal mesmo assim.
Tais "mudanças de contexto" podem ser **chamadas internas a funções**, em linguagens como C; a **métodos**, em linguagens como Java, ou mesmo **chamadas a outras rotinas de outros programas, externos ao programa inicialmente em execução**.
Embora a *stack* esteja dentro do *heap*, sua alocação é estática, ou seja, não mudará durante a execução do programa

Estrutura:

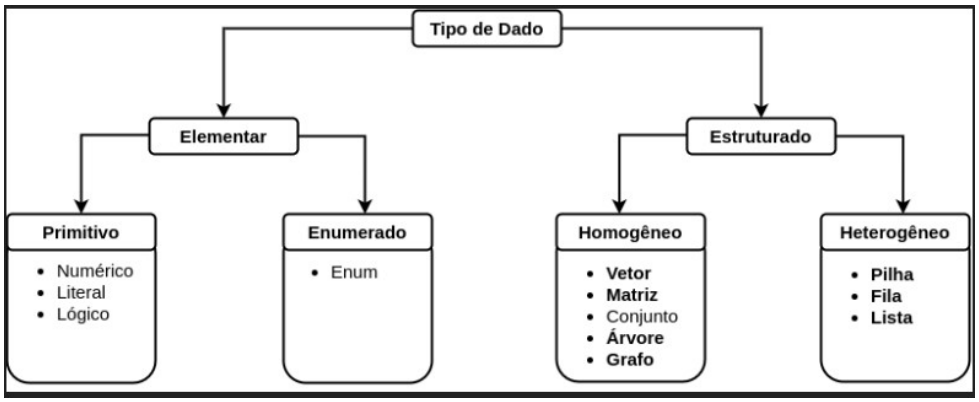


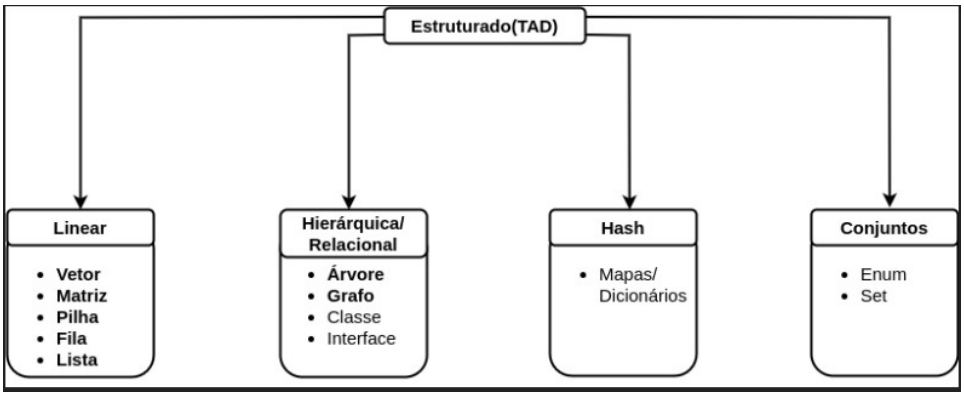


Embora conceitualmente apresentemos o *heap* e *stack* em "caixinhas organizadas", na verdade os dois são criados de forma espaçada (desorganizada) na memória.
Entretanto, o SO tem a capacidade de nos disponibilizar essas caixinhas, para sua compreensão e uso serem mais fáceis.



Heap e Stack funcionando em conjunto. Java


Tipo de Dado
Elementar
Estruturado
Primitivo
Numérico
Literal
Lógico
Enumerado
Enum
Homogêneo
Vetor
Matriz
Conjunto
Árvore
Grafo
Heterogêneo
Pilha
Fila
Lista


Estruturado(TAD)
Linear
Vetor
Matriz
Pilha
Fila
Lista
Hierárquica/ Relacional
Árvore
Grafo
Classe
Interface
Hash
Mapas/ Dicionários
Conjuntos
Enum
Set