


* Array -

- Uma simples estrutura de dados;
- Usados quando temos uma lista de uma sequência de itens, na qual queremos interar os itens um por um ou acessar por meio dos índices.
- Esses itens são armazenados sequencialmente na  memória.
- Usamos uma sequência de números inteiros para acessar os itens do Array.

Ex. `char nome[] = "Palavra";`
`printf("%c", nome[1]);` // exibe na Tela: a

↳ Opera em tempo $O(1)$ constante.

→ Vantagens:

1. Os arrays são ordenados sequencialmente e indexam a localização do itens;
2. Rápido acesso/pesquisa - Você pode obter um item por meio de um índice em tempo $O(1)$ constante;
3. Adiciona e Remove rapidamente no fim - no array é rápido adicionar e remover o último item da lista.

→ Desvantagens:

1. Lento na hora de inserir/deletar - inserir ou deletar um item no meio/início de um array necessita que todos os itens depois dele sejam reindexados;
2. Lentos para buscas - para encontrar um item em um array nos devemos interar/percorrer nossa lista, o que requer um tempo $O(n)$ linear;
3. Um índice indica apenas uma localização em um array, mas não fornece nenhum outro significado sobre o item.

* Como já dito, os arrays armazenam itens sequencialmente na memória.

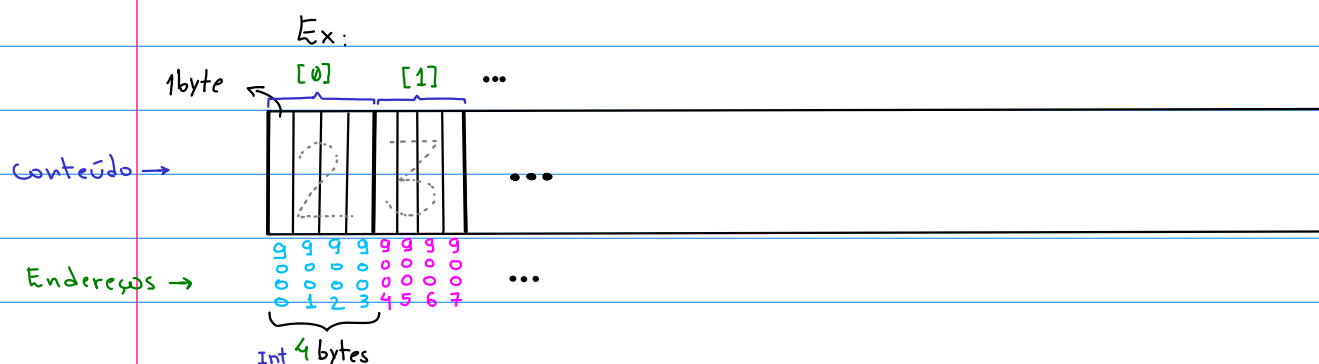
• Esses itens podem ser de qualquer tipo, no entanto cada array deve armazenar seus dados de forma homogênea, isto é, todo o array deve conter um tipo único.

Ex.

2	3	5	7
---	---	---	---

números inteiros

• Quando criamos um array nossa linguagem de programação encontra espaços livres e contínuos na memória.



* Busca em Arrays - $O(n)$

• Nosso array performa-se pobremente quando queremos buscar por um item em particular. Por exemplo, se nós queremos encontrar o 'l' no array char nome[] = "Paulo"; nós teremos que percorrer 3 itens para chegar na consoante 'l'. Em alguns casos precisaremos percorrer todos os n itens.

* Inserir em Arrays - $O(n)$

• Uma das coisas que amamos em arrays é que temos uma lista de itens numerados sequencialmente por meio de índices.

→ O problema disso é que se nós desejamos inserir um elemento no meio ou no início do array vamos ter que "dar um shift" e reindexar cada item que vem após o item que queremos adicionar no array.

Então se nós queremos adicionar um item no começo teremos que reindexar todos os n itens. Assim no pior caso nos podemos dizer que inserir em um array requer uma

complexidade $O(n)$.

Ex.

Quero inserir o 'r' no índice [2], mas já temos um item nessa posição. Logo, só é necessário dar um "shift" em todos os itens que vem depois dele, incluindo até mesmo o atual item na posição [2].

• Array inicial: `char nome[] = { 'w', 'o', 'l', 'd', '\0' };`
[0] [1] [2] [3] [4]
→ → → direita

• Depois: `char nome[] = { 'w', 'o', 'r', 'l', 'd', '\0' };`
[0] [1] [2] [3] [4] [5]
✓ ✓ ✓

x Remover em Arrays - $O(n)$

As remoções são bem semelhantes as inserções. A diferença é que ao remover determinado item, todos os elementos que vem depois dele devem sofrer um "shift" para a esquerda. Pela mesma lógica de adicionar isso resulta em uma operação $O(n)$.

Resumo:

Grande operações
↳ pior caso

			Big O Complexity
		lookup	$O(1)$
0	"Facebook"	set	$O(1)$
1	"Amazon"	search (unsorted)	$O(n)$
2	"Apple"	search (sorted)	$O(\log n)$
3	"Netflix"	insert	$O(n)$
4	"Google"	delete	$O(n)$
5	"Skilled.dev"	append	$O(1)$
6	"gitconnected"	prepend	$O(n)$
		space	$O(n)$

→ Acesso por meio do índice, `nome[3]`.

→ `nome[3] = 'z';`

→ busca com array desordenado.

→ busca com array ordenado

→ Adicionando no meio

→ Remover no início/meio

→ Adicionar no final

→ Adicionar no começo

Array

Acesso	busca - desordenado	adicionar - começo/meio	adicionar - fim
$O(1)$ constante	$O(n)$ Linear	$O(n)$ Linear	$O(1)$ constante
			
Remover - começo/meio	Remover - fim		
$O(n)$ Linear	$O(1)$ constante		
			
buscar - desordenado	buscar - ordenado		
$O(n)$ Linear	$O(\log n)$		
			

-Discente : Paulo Henrique Diniz de Lima Alencar.

Referências:

ARRAYS. Skilled.dev, 2020. Disponível em: <<https://skilled.dev/course/arrays>>. Acesso em: 23 dez. de 2020.