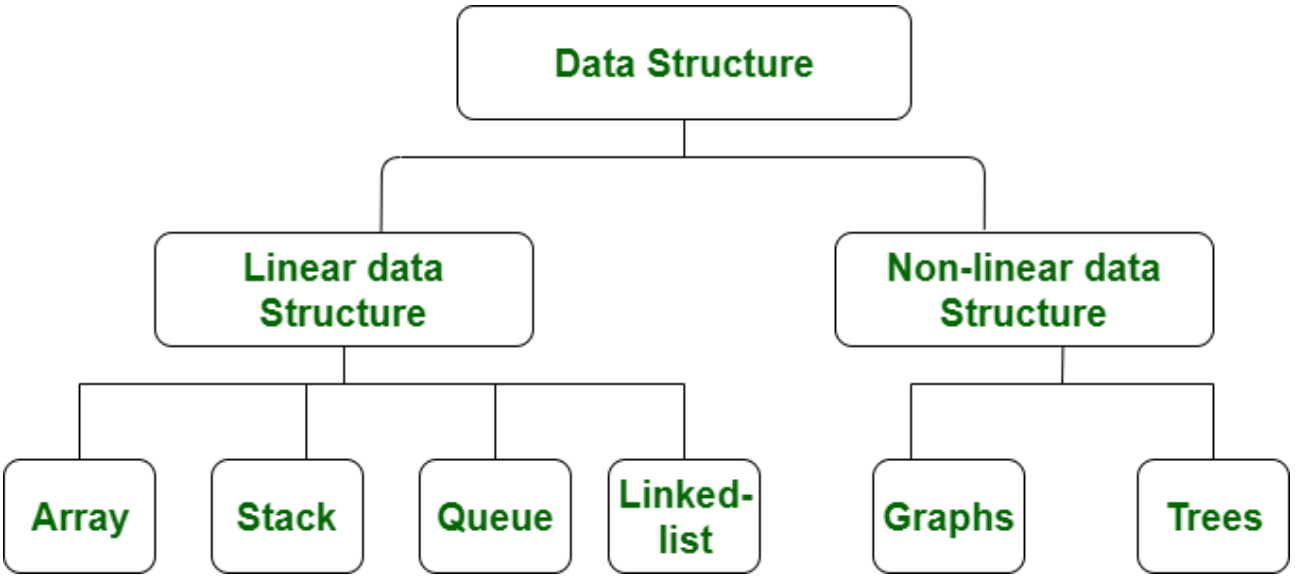
ESTRUTURA DE DADOS AVALIAÇÃO

MÓDULO 1

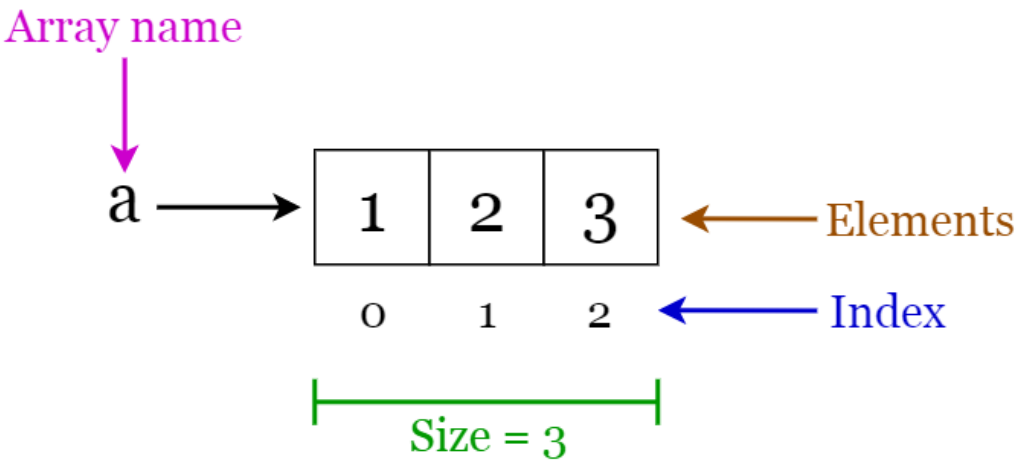
Introdução

Estruturas de dados representam mecanismos de programação pelos quais podemos organizar e armazenar dados de um programa de computador de tal forma que os dados armazenados possam ser manuseados de forma mais eficiente. As estruturas de dados têm um escopo amplo e diversificado de uso nas áreas de Ciência da Computação e Engenharia de Software.



As estruturas de dados são empregadas em quase todos os programas computacionais. Em computação, estruturas de dados são um tópico chave para a formação de qualquer profissional ou pesquisador.

A estrutura de dados mais elementar é certamente o array. Na maioria das linguagens de programação de alto nível, um array possui tamanho fixo, que pode conter itens do mesmo tipo de dados. Um array pode agregar números, strings ou mesmo outros arrays (como matrizes, i. e., arrays bidimensionais).



Como exemplo de operações vinculadas ao manuseio de dados de um array, podemos elencar:

- Percorrer: percorrer os elementos e imprimi-los;
- Pesquisar: buscar um elemento no array;
- Atualizar: atualiza o valor de um elemento existente em um determinado índice.
- Inserção: criar um novo array com tamanho aumentado (tamanho atual + 1), copiar os elementos existentes no array original para o de tamanho aumentado e adicionar o novo elemento ao final do array;
- Remoção: criar um novo array com tamanho inferior (tamanho atual + 1), copiar os elementos existentes no array original para o de tamanho inferior sem considerar o elemento que se quer excluir do array corrente;

Assim como array, existem na literatura outras estruturas de dados cujo projeto é similarmente elementar. Dessas, as mais populares são Pilha, Fila e Lista. O projeto e implementação destas estruturas de dados são tema do nosso próximo módulo. Agora, iremos focar no projeto e implementação de uma estrutura de dados elementar pouco conhecida: Bag (‘saco’ em português).

A implementação dessa estrutura de dados pode ser abstraída em um Tipo Abstrato de Dados que possui as seguintes operações:

- **create**: recebe um inteiro n e cria um array de dimensão um objeto estruturado Bag que possui três atributos: um ponteiro para um array de dimensão **n** dinamicamente alocado o qual irá guardar os dados da estrutura; *n* (ou número máximo de elementos), o qual guarda a dimensão do array de dados para fins de orientar as iterações sobre o vetor; *size* (ou tamanho), o qual guarda o número de inserções feitas no array de dados;
- **insert**: seleciona de forma aleatória uma posição *i*, tal que **0 < *i* < n - 1** e nenhum elemento foi previamente inserido na posição *i*. Insere um elemento *e* na posição *i* do array de dados da respectiva instância de Bag;
- **get**: seleciona de forma aleatória uma posição *i*, tal que **0 < *i* < n - 1** e nesta posição *i* do array de dados da respectiva instância de Bag tenha sido inserido previamente um elemento *e*. O elemento *e* é então retornado e a posição *i* fica vaga para uma nova operação de inserção*;*
- **search:** pesquisa pela ocorrência de um elemento *e* no array de dados da respectiva instância de Bag;
- **size:** retorna o número de elementos inseridos na respectiva instância de Bag;

- **printAll**: imprime todos os elementos inseridos até então na respectiva instância de Bag.

Vamos considerar que nosso array de dados receberá apenas inteiros positivos. O código fonte que se segue apresenta a implementação (incompleta) da estrutura Bag em C:

arquivo bag.h

```
typedef struct bag Bag;

Bag * create(int n);

int insert(Bag* bag, int e);

int get(Bag* bag);

int search(Bag* bag, int e);

int size(Bag* bag);

void printAll(Bag* bag);
```

arquivo bag.c

```
#include <stdio.h>
#include <stdlib.h>
#include "bag.h"
struct bag
{
   int *data;
   int n;
   int size;
};

Bag *create(int n)
{
   Bag *bag = (Bag *)malloc(sizeof(Bag));
   bag->data = (int *)calloc(n, sizeof(int));
   bag->n = n;
   bag->size = 0; // zero, pois nenhum elemento foi inserido.
   printf("Instância de Bag criada em %p!\n\n", bag);
   return bag;
}
```

```c
int insert(Bag *bag, int e)
{
   if (bag->size < bag->n && e > 0)
   { // avalia se bag não está cheio e se o elemento 'e' é positivo
       int i = 0;
       do
       {
           i = rand() % bag->n; // atribui um valor a i, tal que 0 <= i < n
       } while (bag->data[i] != 0); // avalia se em i já não há um elemento
       bag->data[i] = e;
       printf("Elemento %i inserido em bag->[%i].\n\n", e, i);
       return i;
   }
   return -1;
}

int get(Bag *bag)
{
   return 0;
}

int search(Bag *bag, int e)
{
   return 0;
}

int size(Bag *bag)
{
   return 0;
}

void printAll(Bag *bag) {}
```

arquivo main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "bag.h"
int main()
{
```

```
    srand(time(NULL)); //inicializa o gerador de números randômicos.
    Bag *bag = create(2);
    insert(bag, 10);
    insert(bag, 20);
    return 1;
}
```

Desafio

1. Em bag.c, implemente as funções **get, search, size, printAll** conforme foi explicado na descrição das operações dessa estrutura de dados na seção de *Introdução*.

2. Se imagine contratado(a) para programar um software que irá gerar a próxima sequência de números que constituem o resultado da Mega-Sena. Ao invés de usar um simples array para manusear os dados do sorteio, porque poderíamos empregar Bag para auxiliar no procedimento de sortear os números do resultado da Mega-Sena?

As respostas devem vir em um único arquivo no formato PDF. Para tal, você precisará colar o código fonte no seu documento de resposta tal qual está posto o código fonte de implementação parcial de Bag neste documento. A entrega deve ser via SIGAA, somente.