INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO RIO GRANDE DO NORTE
Campus Nova Cruz – Código INEP: 24083003
Av. José Rodrigues de Aquino Filho, Nº 640, RN 120, Alto de Santa Luzia, CEP 59215-000, Nova Cruz (RN)
CNPJ: 10.877.412/0015-633 – Telefone: (84) 4005-4107

Exercício 01 de Implementação:

Estrutura de Dados Array em C++

(EDL – TADS4M)

1. Contextualização do Exercício

Este exercício prático será trabalhado nas primeiras aulas da disciplina de **Estrutura de Dados Lineares**, tendo como objetivo revisar alguns conhecimentos básicos do paradigma de **Programação Orientada a Objetos (POO)** e fornecer o primeiro contato com a linguagem de programação C++. Neste exercício, deve-se implementar uma classe que simula o comportamento de um **vetor dinâmico**, ou seja, uma estrutura de dados que pode aumentar ou diminuir de tamanho em tempo de execução. O foco está na compreensão dos desafios e soluções para gerenciar memória dinamicamente, a correta aplicação de conceitos de POO e a observação na prática de algumas desvantagens em se utilizar um vetor como estrutura de armazenamento de dados, mesmo quando ele é dinâmico.

2. Descrição do Exercício

A classe denominada `Array` deve representar um **vetor dinâmico de inteiros**. A classe deve ser dividida em dois arquivos:

- `Array.hpp`: O arquivo de cabeçalho (interface) que conterá a declaração da classe.
- `Array.cpp`: O arquivo de implementação que conterá a definição dos métodos.

A classe `Array` deve ser capaz de realizar as operações e gerenciar as propriedades que serão listadas nos tópicos seguintes deste documento.

2.1 Atributos:

- `int * dados`: ponteiro para inteiros que deve armazenar os elementos do vetor.
- `int tamanho`: número atual de elementos armazenados no vetor.
- `int capacidade`: capacidade total de armazenamento do vetor.

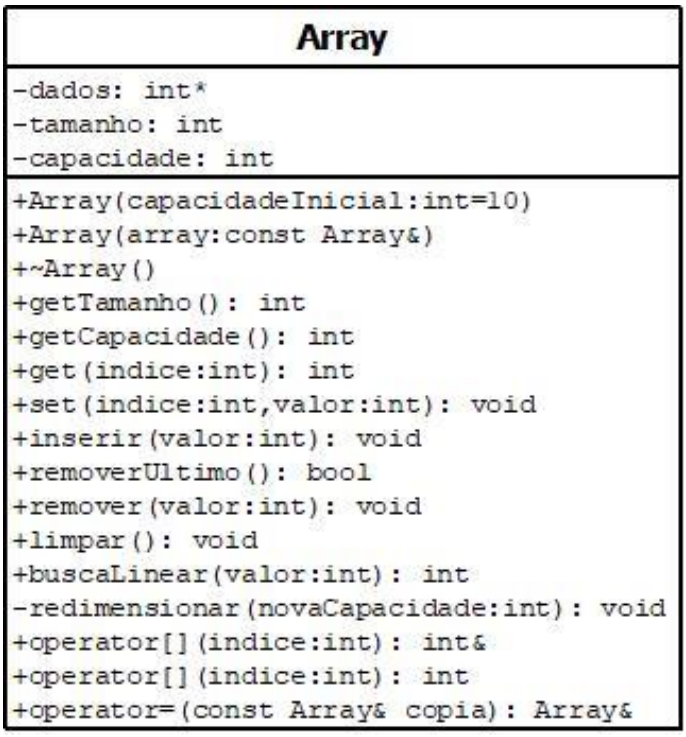
2.2 Método Privado:

- `void redimensionar(int novaCapacidade)`: método auxiliar para realocar o vetor com uma nova capacidade. Este método deve ser chamado internamente sempre que o vetor precisar aumentar de tamanho.

2.3 Métodos Públicos (Interface da Classe):

- **Construtores:**
  - Construtor padrão que recebe a `capacidadeInicial` (com um valor padrão de 10) e inicializa os atributos da classe. Deve lidar com capacidades iniciais inválidas (menores ou iguais a zero).
  - Um **construtor de cópia** que cria um novo objeto idêntico a outro objeto da classe `Array`.
- **Destrutor:**
  - Um destrutor (`~Array()`) para liberar a memória alocada dinamicamente.
- **Métodos Assessores (*getters* e *setters*):**
  - `int getTamanho() const`: retorna o número de elementos no vetor.
  - `int getCapacidade() const`: retorna a capacidade total do vetor.
  - `int get(int indice) const`: retorna o elemento em um `indice` específico. Deve lançar uma exceção se o índice for inválido.
  - `void set(int indice, int valor)`: define um novo valor em um `indice` específico. Deve lançar uma exceção se o índice for inválido.
- **Manipulação de Elementos:**
  - `void inserir(int valor)`: adiciona um novo elemento ao final do vetor. Se a capacidade for atingida, o vetor deve ser redimensionado (dobre sua capacidade).
  - `bool removerUltimo()`: remove o último elemento do vetor, se ele não estiver vazio.
  - `void remover(int indice)`: remove o elemento de um `indice` específico. Os elementos subsequentes devem ser movidos para preencher o espaço. Deve lançar uma exceção se o índice for inválido.
  - `void limpar()`: remove todos os elementos do vetor, redefinindo o seu `tamanho` para zero.

- **Busca:**
  - `int buscaLinear(int valor) const`: realiza uma busca linear no vetor para encontrar um valor, retornando o valor de seu índice. Caso o valor não seja encontrado, o método deve retornar -1.
- **Sobrecarga de Operadores:**
  - `int & operator[](int indice)` e `int operator[](int indice) const`: sobrecarga do operador [] para permitir a leitura e a escrita em um índice específico do `Array`, de forma similar a um vetor nativo.
  - `const Array & operator=(const Array & direita)`: sobrecarga do operador de atribuição = para permitir que um `Array` seja copiado para outro.
- **Utilitários:**
  - `void imprimir() const`: imprime em tela todos os elementos do objeto da classe `Array`.



3. Funcionalidades adicionais a serem implementadas

A descrição inicial da classe `Array` está prevista para ser trabalhada no Laboratório de Informática. Assim, à medida que o código vai sendo desenvolvido serão apresentadas explicações conceituais de estruturas de dados, programação orientada a objetos e específicas da linguagem de programação C++. Nesta seção são apresentadas funcionalidades adicionais que você deve implementar na classe `Array` como forma de praticar o que foi visto durante as aulas:

1. **Inserir no Início:** crie um método chamado `inserirNoInicio(int valor)` que insere um novo elemento no início do `Array`. Para isso, todos os elementos existentes precisarão ser deslocados para a direita. Lembre-se de verificar se o `Array` precisa ser redimensionado antes da inserção.
2. **Remover do Início:** crie um método chamado `removerPrimeiro()` que remove o primeiro elemento do `Array`. Desloque todos os elementos restantes para a esquerda para preencher o espaço vazio. O método deve retornar um `bool` indicando se a remoção foi bem-sucedida.
3. **Inserir em Posição Específica:** crie um método `inserir(int indice, int valor)` que insere um elemento em uma posição específica do `Array`. Certifique-se de validar o índice fornecido.
4. **Busca Binária:** implemente o método `buscaBinaria(int valor)` que executa o algoritmo de busca binária para encontrar um valor no `Array`.
    - **Observação:** a busca binária só funciona em vetores ordenados. Implemente o método `ordenar()` que executa algum dos algoritmos tradicionais de ordenação de vetores (*Bubble Sort*, *Insertion Sort*, *Selection Sort*, *Merge Sort*, entre outros) para ordenar os elementos do objeto da classe `Array`;
    - Crie o método estático `Array ordenar(const Array & array)` que retorna uma cópia ordenada do parâmetro array, com elementos ordenados pela execução de algum dos algoritmos tradicionais de ordenação de vetores (o mesmo implementado no método `ordenar`).

4. O que vem por aí?

Na implementação da classe `Array`, consideramos algumas características peculiares que foram disponibilizadas por meio de sua interface. Por exemplo, o método `inserir` adiciona elementos ao final do vetor de elementos, enquanto o método `removerUltimo` remove o último elemento armazenado na estrutura de dados. Na seção anterior, sugeriu-se a implementação do método `removerPrimeiro`, que remove o primeiro elemento armazenado no objeto da classe `Array`. Neste contexto, nota-se que é possível ter estruturas levemente diferentes tendo a mesma base de armazenamento:

- Considere, por exemplo, que em um `Array` os elementos são incluídos sempre ao final do vetor de dados e removidos sempre do início. Essa é uma estrutura em que o primeiro elemento a entrar é o primeiro elemento a sair (estrutura FIFO, do inglês *First-In*, *First-Out*, ou Primeiro a Entrar, Primeiro a Sair). Que situação de nosso dia a dia se assemelha a essa estrutura?
- Considere uma outra situação em que os elementos de um `Array` em que tanto a adição quanto a remoção de elementos são realizadas no final do vetor de dados. Essa é uma estrutura em que o

último elemento a entrar é o primeiro elemento a sair (estrutura LIFO, do inglês *Last-In*, *First-Out*, ou Último a Entrar, Primeiro a Sair). Que aspecto da execução de um programa de computador baseado em chamadas de funções/métodos se assemelha ao comportamento dessa estrutura?