Iteradores

Estrutura de Dados — QXD0010

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará
Campus Quixadá

2º semestre/2023

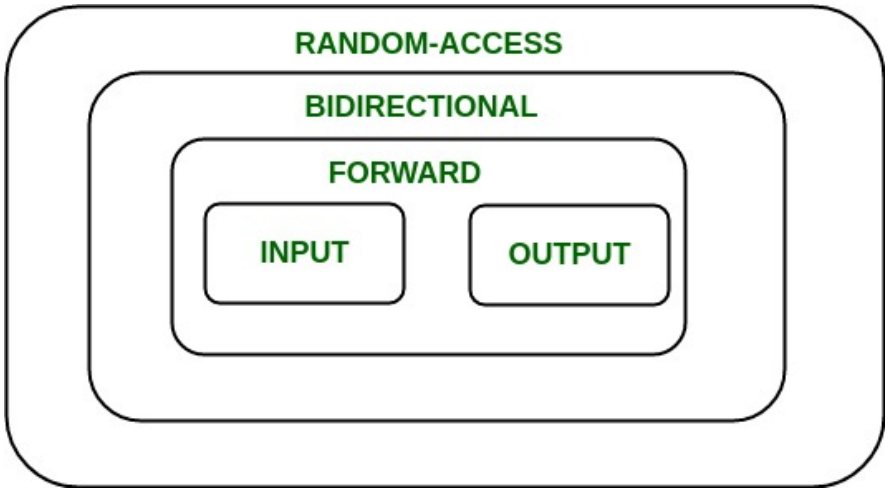
Iteradores

- Um iterador é um objeto que fornece uma maneira de acessar e iterar sobre uma coleção de elementos.
  - atua como um ponteiro que aponta para um elemento da coleção e fornece uma maneira de passar de um elemento para o próximo

Iteradores

- Em C++, os iteradores são classificados em cinco categorias: Input Iterator, Output Iterator, Forward Iterator, Bidirectional Iterator and Random Access Iterator.

- Cada tipo de iterador fornece um nível diferente de funcionalidade e oferece suporte a diferentes operações.



Input Iterator

- Input Iterators permitem percorrer a coleção do início ao fim acessando os elementos em modo somente leitura.
- Só podem ser incrementados e não suportam decrementação ou acesso aleatório.

Input Iterator

- Input Iterators permitem percorrer a coleção do início ao fim acessando os elementos em modo somente leitura.
- Só podem ser incrementados e não suportam decrementação ou acesso aleatório.
- Ele precisa implementar:
  - o operador de pré-incremento: operator++()
  - o operador de desrreferencia: operator*()
  - e os operadores de igualdade: operator==() e operator!=()

Output Iterator

- Output Iterator fornece acesso somente gravação a uma coleção e permite percorrer a coleção do início ao fim.
- Só podem ser incrementados e não suportam decrementação ou acesso aleatório.

Output Iterator

- Output Iterator fornece acesso somente gravação a uma coleção e permite percorrer a coleção do início ao fim.
- Só podem ser incrementados e não suportam decrementação ou acesso aleatório.
- Ele precisa implementar:
  - o operador de pré-incremento: operator++()
  - o operador de desrreferencia: operator*()
  - e os operadores de atribuição: operator=()

Forward Iterator

- Este tipo de iterador permite percorrer a coleção para frente, seja para ler ou escrever os elementos.
- Um Forward Iterator precisa suportar as mesmas operações que um Input Iterator, mais o operador pós-incremento operador++(int).
- Forward Iterators não suportam decrementação ou acesso aleatório.

Bidirectional Iterator

- Este tipo de iterador permite percorrer a coleção tanto para frente quanto para trás, seja para ler ou escrever os elementos.

- Um Bidirectional Iterator precisa suportar todas as operações de um Forward Iterator, mais os seguintes operadores:
  - o operador de pré-decremento operator−−()
  - e o operador de pós-decremento operator−−(int)

Random Access Iterator

- Esse tipo de iterador oferece maior funcionalidade e permite acessar os elementos da coleção em qualquer ordem.

Random Access Iterator

- Esse tipo de iterador oferece maior funcionalidade e permite acessar os elementos da coleção em qualquer ordem.
- Ele precisa suportar todas as operações de um Bidirectional Iterator, mais:
  - os operadores aritméticos: operator+(), operator−(), operator+=(), operator−=()
  - o operador de indexação: operator[ ]()
  - e os operadores de comparação: operator<(), operator>(), operator<=() e operator>=()

Um iterador para Vector

Um iterador para a nossa classe Vector

- Vamos programar uma classe `iterator` para o nosso Vector genérico.
- Vamos programar `iterator` como uma classe interna da classe Vector.

Um iterador para a nossa classe Vector

- Vamos programar uma classe `iterator` para o nosso Vector genérico.
- Vamos programar `iterator` como uma classe interna da classe Vector.
- Nossa classe iterator será do tipo Random Access Iterator
- Ou seja, com esse iterador será possível caminhar pelo Vector para trás ou para frente, ou até mesmo dar saltos.

Funções da nossa classe `iterator`

A fim de programar a classe `iterator` como um Random Access Iterator, como vimos, precisaremos sobrecarregar vários operadores. A seguir, segue uma lista de todos eles.

Funções da nossa classe `iterator`

A fim de programar a classe `iterator` como um Random Access Iterator, como vimos, precisaremos sobrecarregar vários operadores. A seguir, segue uma lista de todos eles.

- As 4 operações criadas por padrão pelo C++ não precisam ser sobrescritas. As versões fornecidas pelo C++ funcionam corretamente com o nosso iterador.
  - Construtor default
  - Construtor de cópia
  - Destrutor
  - operador de atribuição

Funções da nossa classe `iterator`

- `iterator(T *ptr)`
  Construtor customizado. Recebe um ponteiro para um elemento do array e cria um *iterator* apontando para esse elemento.

Funções da nossa classe `iterator`

- `iterator(T *ptr)`
  Construtor customizado. Recebe um ponteiro para um elemento do array e cria um *iterator* apontando para esse elemento.

- `T& operator*()`
  Operador de desreferência. Retorna uma referência para o elemento apontado pelo *iterator*.

Funções da nossa classe `iterator`

- `iterator(T *ptr)`
  Construtor customizado. Recebe um ponteiro para um elemento do array e cria um *iterator* apontando para esse elemento.

- `T& operator*()`
  Operador de desreferência. Retorna uma referência para o elemento apontado pelo *iterator*.

- `T* operator->()`
  Operador seta. Retorna o valor do atributo privado do *iterator*, que é um ponteiro.

Funções da nossa classe `iterator`

- `iterator(T *ptr)`
  Construtor customizado. Recebe um ponteiro para um elemento do array e cria um *iterator* apontando para esse elemento.

- `T& operator*()`
  Operador de desreferência. Retorna uma referência para o elemento apontado pelo *iterator*.

- `T* operator->()`
  Operador seta. Retorna o valor do atributo privado do *iterator*, que é um ponteiro.

- `T& operator[](long n)`
  Operador de indexação. Retorna uma referência para o elemento localizado n posições após a posição atual do *iterator*.

Funções da nossa classe `iterator`

- `iterator& operator++()`
  Operador de pré-incremento. Permite expressões do tipo `++it`

Funções da nossa classe `iterator`

- `iterator& operator++()`
  Operador de pré-incremento. Permite expressões do tipo `++it`

- `iterator operator++(int)`
  Operador de pós-incremento. Permite expressões do tipo `it++`

Funções da nossa classe `iterator`

- `iterator& operator++()`
  Operador de pré-incremento. Permite expressões do tipo `++it`

- `iterator operator++(int)`
  Operador de pós-incremento. Permite expressões do tipo `it++`

- `iterator& operator--()`
  Operador de pré-decremento. Permite expressões do tipo `--it`

Funções da nossa classe `iterator`

- `iterator& operator++()`
  Operador de pré-incremento. Permite expressões do tipo `++it`

- `iterator operator++(int)`
  Operador de pós-incremento. Permite expressões do tipo `it++`

- `iterator& operator--()`
  Operador de pré-decremento. Permite expressões do tipo `--it`

- `iterator operator--(int)`
  Operador de pós-decremento. Permite expressões do tipo `it--`

Funções da nossa classe `iterator`

- `iterator& operator++()`
  Operador de pré-incremento. Permite expressões do tipo `++it`

- `iterator operator++(int)`
  Operador de pós-incremento. Permite expressões do tipo `it++`

- `iterator& operator--()`
  Operador de pré-decremento. Permite expressões do tipo `--it`

- `iterator operator--(int)`
  Operador de pós-decremento. Permite expressões do tipo `it--`

Funções da nossa classe `iterator`

- `iterator& operator+=(long n)`
  Operador de soma e atribuição.
  Permite expressões do tipo `it += 5`

Funções da nossa classe `iterator`

- `iterator& operator+=(long n)`
  Operador de soma e atribuição.
  Permite expressões do tipo `it += 5`

- `iterator& operator-=(long n)`
  Operador de subtração e atribuição.
  Permite expressões do tipo `it -= 5`

Funções da nossa classe `iterator`

Operadores relacionais

Vamos implementar como funções globais amigas:

- `bool operator==(const iterator& l, const iterator& r)`

Funções da nossa classe `iterator`

Operadores relacionais

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Vamos implementar como funções globais amigas:

- `bool operator==(const iterator& l, const iterator& r)`
- `bool operator!=(const iterator& l, const iterator& r)`

Funções da nossa classe `iterator`

Operadores relacionais

Vamos implementar como funções globais amigas:

- `bool operator==(const iterator& l, const iterator& r)`
- `bool operator!=(const iterator& l, const iterator& r)`
- `bool operator<(const iterator& l, const iterator& r)`

Funções da nossa classe `iterator`

Operadores relacionais

Vamos implementar como funções globais amigas:

- `bool operator>(const iterator& l, const iterator& r)`

Funções da nossa classe `iterator`

Operadores relacionais

Vamos implementar como funções globais amigas:

- `bool operator>(const iterator& l, const iterator& r)`
- `bool operator<=(const iterator& l, const iterator& r)`

Funções da nossa classe `iterator`

Operadores relacionais

Vamos implementar como funções globais amigas:

- `bool operator>(const iterator& l, const iterator& r)`
- `bool operator<=(const iterator& l, const iterator& r)`
- `bool operator>=(const iterator& l, const iterator& r)`

Funções da nossa classe `iterator`

- `friend iterator operator+(const iterator& it, long n)`
  Operador de soma com inteiro. Permite expressões do tipo `it + 5`.
  Retorna um novo iterador apontando n posições à frente de `it`.

Funções da nossa classe `iterator`

- `friend iterator operator+(const iterator& it, long n)`
  Operador de soma com inteiro. Permite expressões do tipo `it + 5`.
  Retorna um novo iterador apontando n posições à frente de `it`.

- `friend iterator operator-(const iterator& it, long n)`
  Operador de soma com inteiro. Permite expressões do tipo `it - 5`.
  Retorna um novo iterador apontando n posições atrás de `it`.

Funções da nossa classe `iterator`

- `friend iterator operator+(const iterator& it, long n)`
  Operador de soma com inteiro. Permite expressões do tipo `it + 5`.
  Retorna um novo iterador apontando n posições à frente de `it`.

- `friend iterator operator-(const iterator& it, long n)`
  Operador de soma com inteiro. Permite expressões do tipo `it - 5`.
  Retorna um novo iterador apontando n posições atrás de `it`.

- `friend long operator-(const iterator& l, const iterator& r)`
  Operador de subtração entre interators. Retorna a diferença entre os ponteiros dos dois iteradores.
  Permite operações do tipo `it1 - it2`

Funções da classe Vector

Por fim, a classe Vector deve ter duas funções que retornam um iterator para a primeira posição e outra com um iterator para a última posição.

- `iterator begin()`
- `iterator end()`

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM