Estrutura de Dados

Paulo Torrens

paulotorrens@gnu.org

Departamento de Ciência da Computação
Centro de Ciências e Tecnológias
Universidade do Estado de Santa Catarina

2020/1

Coleções Dinâmicas

- Variáveis dentro de um programa são classificadas pelo seu **tipo**: inteiros, caracteres, *strings*...
- Frequentemente em programas há a necessidade de se salvar **coleções** (do inglês, *containers*) de dados em uma única variável (e.g., um vetor de inteiros, uma lista de usuários, etc)
- Muitas vezes, essas coleções precisam ser feitas de forma **dinâmica** dentro de um sistema
  - Não se sabe quantos objetos existirão dentro da coleção em tempo de compilação, ou o número de objetos será alterado durante a execução do programa
  - Precisamos da capacidade de **adicionar** e **remover** objetos dessa coleção
  - Em linguagens como C, tais coleções podem ser representados na forma de **tipos de dados abstratos**

Coleções Dinâmicas

- Variáveis dentro de um programa são classificadas pelo seu **tipo**: inteiros, caracteres, *strings*...
- Frequentemente em programas há a necessidade de se salvar **coleções** (do inglês, *containers*) de dados em uma única variável (e.g., um vetor de inteiros, uma lista de usuários, etc)
- Muitas vezes, essas coleções precisam ser feitas de forma **dinâmica** dentro de um sistema
  - Não se sabe quantos objetos existirão dentro da coleção em tempo de compilação, ou o número de objetos será alterado durante a execução do programa
  - Precisamos da capacidade de **adicionar** e **remover** objetos dessa coleção
  - Em linguagens como C, tais coleções podem ser representados na forma de **tipos de dados abstratos**

Coleções Dinâmicas

- Variáveis dentro de um programa são classificadas pelo seu **tipo**: inteiros, caracteres, *strings*...
- Frequentemente em programas há a necessidade de se salvar **coleções** (do inglês, *containers*) de dados em uma única variável (e.g., um vetor de inteiros, uma lista de usuários, etc)
- Muitas vezes, essas coleções precisam ser feitas de forma **dinâmica** dentro de um sistema
  - Não se sabe quantos objetos existirão dentro da coleção em tempo de compilação, ou o número de objetos será alterado durante a execução do programa
  - Precisamos da capacidade de **adicionar** e **remover** objetos dessa coleção
  - Em linguagens como C, tais coleções podem ser representados na forma de **tipos de dados abstratos**

Coleções Dinâmicas

- Variáveis dentro de um programa são classificadas pelo seu **tipo**: inteiros, caracteres, *strings*...
- Frequentemente em programas há a necessidade de se salvar **coleções** (do inglês, *containers*) de dados em uma única variável (e.g., um vetor de inteiros, uma lista de usuários, etc)
- Muitas vezes, essas coleções precisam ser feitas de forma **dinâmica** dentro de um sistema
  - Não se sabe quantos objetos existirão dentro da coleção em tempo de compilação, ou o número de objetos será alterado durante a execução do programa
  - Precisamos da capacidade de **adicionar** e **remover** objetos dessa coleção
  - Em linguagens como C, tais coleções podem ser representados na forma de **tipos de dados abstratos**

Coleções Dinâmicas

- Variáveis dentro de um programa são classificadas pelo seu **tipo**: inteiros, caracteres, *strings*...
- Frequentemente em programas há a necessidade de se salvar **coleções** (do inglês, *containers*) de dados em uma única variável (e.g., um vetor de inteiros, uma lista de usuários, etc)
- Muitas vezes, essas coleções precisam ser feitas de forma **dinâmica** dentro de um sistema
  - Não se sabe quantos objetos existirão dentro da coleção em tempo de compilação, ou o número de objetos será alterado durante a execução do programa
  - Precisamos da capacidade de **adicionar** e **remover** objetos dessa coleção
  - Em linguagens como C, tais coleções podem ser representados na forma de **tipos de dados abstratos**

Coleções Dinâmicas

- Variáveis dentro de um programa são classificadas pelo seu **tipo**: inteiros, caracteres, *strings*...
- Frequentemente em programas há a necessidade de se salvar **coleções** (do inglês, *containers*) de dados em uma única variável (e.g., um vetor de inteiros, uma lista de usuários, etc)
- Muitas vezes, essas coleções precisam ser feitas de forma **dinâmica** dentro de um sistema
  - Não se sabe quantos objetos existirão dentro da coleção em tempo de compilação, ou o número de objetos será alterado durante a execução do programa
  - Precisamos da capacidade de **adicionar** e **remover** objetos dessa coleção
  - Em linguagens como C, tais coleções podem ser representados na forma de **tipos de dados abstratos**

Filas, pilhas e listas

- Dentre as coleções clássica estudadas, podemos citas as **filas** e as **pilhas**
- Uma fila representa uma estrutura de dados **FIFO** (do inglês, *first-in, first-out*)
  - Podemos **enfileirar** um objeto (*enqueue*), o adicionando no **fim** da fila
  - E podemos **desenfileirar** um objeto (*dequeue*), removendo o item no **início** da fila
- Uma pilha representa uma estrutura de dados **LIFO** (do inglês, *last-in, first-out*)
  - Podemos **empilhar** um objeto (*push*), o adicionando no **topo** da pilha
  - E podemos **desempilhar** um objeto (*pop*), removendo o item do **tipo** da pilha

Filas, pilhas e listas

- Dentre as coleções clássica estudadas, podemos citas as **filas** e as **pilhas**
- Uma fila representa uma estrutura de dados **FIFO** (do inglês, *first-in, first-out*)
  - Podemos **enfileirar** um objeto (*enqueue*), o adicionando no **fim** da fila
  - E podemos **desenfileirar** um objeto (*dequeue*), removendo o item no **início** da fila
- Uma pilha representa uma estrutura de dados **LIFO** (do inglês, *last-in, first-out*)
  - Podemos **empilhar** um objeto (*push*), o adicionando no **topo** da pilha
  - E podemos **desempilhar** um objeto (*pop*), removendo o item do **tipo** da pilha

Filas, pilhas e listas

- Dentre as coleções clássica estudadas, podemos citas as **filas** e as **pilhas**
- Uma fila representa uma estrutura de dados **FIFO** (do inglês, *first-in, first-out*)
  - Podemos **enfileirar** um objeto (*enqueue*), o adicionando no **fim** da fila
  - E podemos **desenfileirar** um objeto (*dequeue*), removendo o item no **início** da fila
- Uma pilha representa uma estrutura de dados **LIFO** (do inglês, *last-in, first-out*)
  - Podemos **empilhar** um objeto (*push*), o adicionando no **topo** da pilha
  - E podemos **desempilhar** um objeto (*pop*), removendo o item do **tipo** da pilha

Filas, pilhas e listas

- Dentre as coleções clássica estudadas, podemos citas as **filas** e as **pilhas**
- Uma fila representa uma estrutura de dados **FIFO** (do inglês, *first-in, first-out*)
  - Podemos **enfileirar** um objeto (*enqueue*), o adicionando no **fim** da fila
  - E podemos **desenfileirar** um objeto (*dequeue*), removendo o item no **início** da fila
- Uma pilha representa uma estrutura de dados **LIFO** (do inglês, *last-in, first-out*)
  - Podemos **empilhar** um objeto (*push*), o adicionando no **topo** da pilha
  - E podemos **desempilhar** um objeto (*pop*), removendo o item do **tipo** da pilha

Filas, pilhas e listas

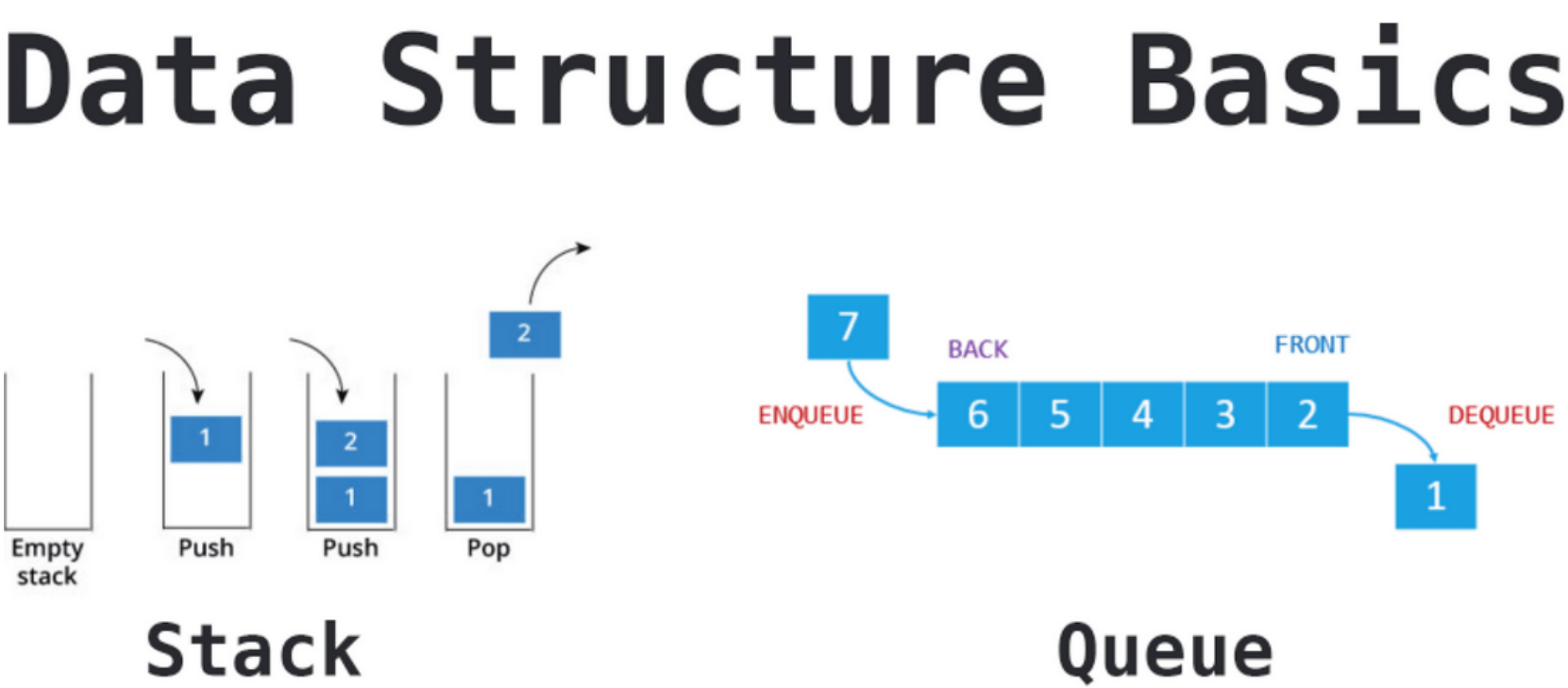
- Dentre as coleções clássica estudadas, podemos citas as **filas** e as **pilhas**
- Uma fila representa uma estrutura de dados **FIFO** (do inglês, *first-in, first-out*)
  - Podemos **enfileirar** um objeto (*enqueue*), o adicionando no **fim** da fila
  - E podemos **desenfileirar** um objeto (*dequeue*), removendo o item no **início** da fila
- Uma pilha representa uma estrutura de dados **LIFO** (do inglês, *last-in, first-out*)
  - Podemos **empilhar** um objeto (*push*), o adicionando no **topo** da pilha
  - E podemos **desempilhar** um objeto (*pop*), removendo o item do **tipo** da pilha

Filas, pilhas e listas

- Dentre as coleções clássica estudadas, podemos citas as **filas** e as **pilhas**
- Uma fila representa uma estrutura de dados **FIFO** (do inglês, *first-in, first-out*)
  - Podemos **enfileirar** um objeto (*enqueue*), o adicionando no **fim** da fila
  - E podemos **desenfileirar** um objeto (*dequeue*), removendo o item no **início** da fila
- Uma pilha representa uma estrutura de dados **LIFO** (do inglês, *last-in, first-out*)
  - Podemos **empilhar** um objeto (*push*), o adicionando no **topo** da pilha
  - E podemos **desempilhar** um objeto (*pop*), removendo o item do **tipo** da pilha

Filas, pilhas e listas

- Dentre as coleções clássica estudadas, podemos citas as **filas** e as **pilhas**
- Uma fila representa uma estrutura de dados **FIFO** (do inglês, *first-in, first-out*)
  - Podemos **enfileirar** um objeto (*enqueue*), o adicionando no **fim** da fila
  - E podemos **desenfileirar** um objeto (*dequeue*), removendo o item no **início** da fila
- Uma pilha representa uma estrutura de dados **LIFO** (do inglês, *last-in, first-out*)
  - Podemos **empilhar** um objeto (*push*), o adicionando no **topo** da pilha
  - E podemos **desempilhar** um objeto (*pop*), removendo o item do **tipo** da pilha


Data Structure Basics
Empty stack
Push
Push
Pop
2
1
2
1
1
Stack
7
BACK
FRONT
ENQUEUE
6
5
4
3
2
DEQUEUE
1
Queue

Listas duplamente linkadas

- Uma das formas de se implementar filas e pilhas é através de listas linkadas
  - Porém, como as estruturas são **abstratas**, não é a única forma: por exemplo, vetores dinâmicos e listas circulares podem ser utilizados
- A ideia de uma lista linkada é representar os dados como uma **corrente** através de elos, pequenas estruturas contendo um valor que pertence à coleção
  - Elos salvam uma referência para o **próximo** elo da corrente
  - Além disso, a estrutura se lembra quem é seu primeiro elo, chamado de **cabeça**
  - Caso cada elo **também** se lembre do elo **anterior**, podemos nos mover em ambas as direções
  - Nesse caso, a lista também deverá se lembrar de sua **cauda**, e teremos uma lista **duplamente encadeada**

Listas duplamente linkadas

- Uma das formas de se implementar filas e pilhas é através de listas linkadas
  - Porém, como as estruturas são **abstratas**, não é a única forma: por exemplo, vetores dinâmicos e listas circulares podem ser utilizados
- A ideia de uma lista linkada é representar os dados como uma **corrente** através de elos, pequenas estruturas contendo um valor que pertence à coleção
  - Elos salvam uma referência para o **próximo** elo da corrente
  - Além disso, a estrutura se lembra quem é seu primeiro elo, chamado de **cabeça**
  - Caso cada elo **também** se lembre do elo **anterior**, podemos nos mover em ambas as direções
  - Nesse caso, a lista também deverá se lembrar de sua **cauda**, e teremos uma lista **duplamente encadeada**

Listas duplamente linkadas

- Uma das formas de se implementar filas e pilhas é através de listas linkadas
  - Porém, como as estruturas são **abstratas**, não é a única forma: por exemplo, vetores dinâmicos e listas circulares podem ser utilizados
- A ideia de uma lista linkada é representar os dados como uma **corrente** através de elos, pequenas estruturas contendo um valor que pertence à coleção
  - Elos salvam uma referência para o **próximo** elo da corrente
  - Além disso, a estrutura se lembra quem é seu primeiro elo, chamado de **cabeça**
  - Caso cada elo **também** se lembre do elo **anterior**, podemos nos mover em ambas as direções
  - Nesse caso, a lista também deverá se lembrar de sua **cauda**, e teremos uma lista **duplamente encadeada**

Listas duplamente linkadas

- Uma das formas de se implementar filas e pilhas é através de listas linkadas
  - Porém, como as estruturas são **abstratas**, não é a única forma: por exemplo, vetores dinâmicos e listas circulares podem ser utilizados
- A ideia de uma lista linkada é representar os dados como uma **corrente** através de elos, pequenas estruturas contendo um valor que pertence à coleção
  - Elos salvam uma referência para o **próximo** elo da corrente
  - Além disso, a estrutura se lembra quem é seu primeiro elo, chamado de **cabeça**
  - Caso cada elo **também** se lembre do elo **anterior**, podemos nos mover em ambas as direções
  - Nesse caso, a lista também deverá se lembrar de sua **cauda**, e teremos uma lista **duplamente encadeada**

Listas duplamente linkadas

- Uma das formas de se implementar filas e pilhas é através de listas linkadas
  - Porém, como as estruturas são **abstratas**, não é a única forma: por exemplo, vetores dinâmicos e listas circulares podem ser utilizados
- A ideia de uma lista linkada é representar os dados como uma **corrente** através de elos, pequenas estruturas contendo um valor que pertence à coleção
  - Elos salvam uma referência para o **próximo** elo da corrente
  - Além disso, a estrutura se lembra quem é seu primeiro elo, chamado de **cabeça**
  - Caso cada elo **também** se lembre do elo **anterior**, podemos nos mover em ambas as direções
  - Nesse caso, a lista também deverá se lembrar de sua **cauda**, e teremos uma lista **duplamente encadeada**

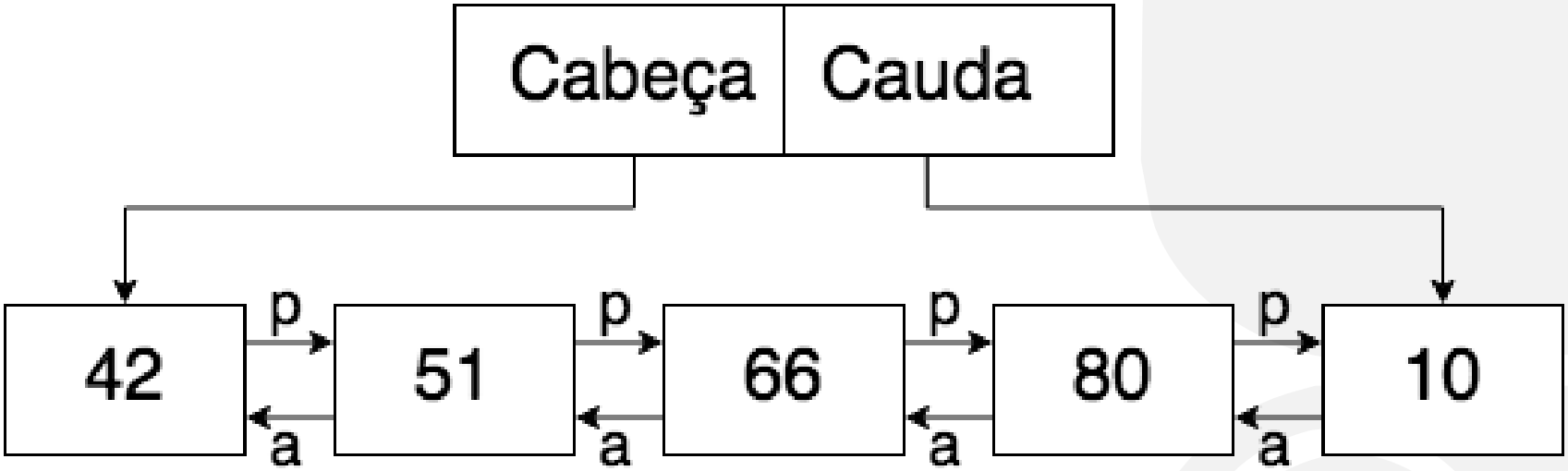
Listas duplamente linkadas

- Uma das formas de se implementar filas e pilhas é através de listas linkadas
  - Porém, como as estruturas são **abstratas**, não é a única forma: por exemplo, vetores dinâmicos e listas circulares podem ser utilizados
- A ideia de uma lista linkada é representar os dados como uma **corrente** através de elos, pequenas estruturas contendo um valor que pertence à coleção
  - Elos salvam uma referência para o **próximo** elo da corrente
  - Além disso, a estrutura se lembra quem é seu primeiro elo, chamado de **cabeça**
  - Caso cada elo **também** se lembre do elo **anterior**, podemos nos mover em ambas as direções
  - Nesse caso, a lista também deverá se lembrar de sua **cauda**, e teremos uma lista **duplamente encadeada**

Listas duplamente linkadas

- Uma das formas de se implementar filas e pilhas é através de listas linkadas
  - Porém, como as estruturas são **abstratas**, não é a única forma: por exemplo, vetores dinâmicos e listas circulares podem ser utilizados
- A ideia de uma lista linkada é representar os dados como uma **corrente** através de elos, pequenas estruturas contendo um valor que pertence à coleção
  - Elos salvam uma referência para o **próximo** elo da corrente
  - Além disso, a estrutura se lembra quem é seu primeiro elo, chamado de **cabeça**
  - Caso cada elo **também** se lembre do elo **anterior**, podemos nos mover em ambas as direções
  - Nesse caso, a lista também deverá se lembrar de sua **cauda**, e teremos uma lista **duplamente encadeada**

Listas duplamente linkadas



Vetores Dinâmicos

- Vetores são áreas contínuas de memória, cujo principal objetivo é o acesso randômico: itens podem ser verificados em qualquer posição de forma rápida
- Entretanto, o tamanho de vetores é fixo: não podemos adicionar, arbitrariamente, mais elementos
- Um vetor dinâmico é uma estrutura de dados abstrata definida a fim de se gerenciar um vetor internamente, podendo **redimensioná-lo** caso necessário
  - Para tal, um vetor dinâmico deve lembrar do endereço do vetor interno, do seu tamanho, e sua **capacidade**
- Além da vantagem de acesso randômico, é possível para um vetor dinâmico fornecer ao usuário do código o endereço para o **armazenamento interno**
  - Entretanto, **remover itens** de um vetor, exceto que na última posição, é **muito custoso**!

Vetores Dinâmicos

- Vetores são áreas contínuas de memória, cujo principal objetivo é o acesso randômico: itens podem ser verificados em qualquer posição de forma rápida
- Entretanto, o tamanho de vetores é fixo: não podemos adicionar, arbitrariamente, mais elementos
- Um vetor dinâmico é uma estrutura de dados abstrata definida a fim de se gerenciar um vetor internamente, podendo **redimensioná-lo** caso necessário
  - Para tal, um vetor dinâmico deve lembrar do endereço do vetor interno, do seu tamanho, e sua **capacidade**
- Além da vantagem de acesso randômico, é possível para um vetor dinâmico fornecer ao usuário do código o endereço para o **armazenamento interno**
  - Entretanto, **remover itens** de um vetor, exceto que na última posição, é **muito custoso**!

Vetores Dinâmicos

- Vetores são áreas contínuas de memória, cujo principal objetivo é o acesso randômico: itens podem ser verificados em qualquer posição de forma rápida
- Entretanto, o tamanho de vetores é fixo: não podemos adicionar, arbitrariamente, mais elementos
- Um vetor dinâmico é uma estrutura de dados abstrata definida a fim de se gerenciar um vetor internamente, podendo **redimensioná-lo** caso necessário
  - Para tal, um vetor dinâmico deve lembrar do endereço do vetor interno, do seu tamanho, e sua **capacidade**
- Além da vantagem de acesso randômico, é possível para um vetor dinâmico fornecer ao usuário do código o endereço para o **armazenamento interno**
  - Entretanto, **remover itens** de um vetor, exceto que na última posição, é **muito custoso**!

Vetores Dinâmicos

- Vetores são áreas contínuas de memória, cujo principal objetivo é o acesso randômico: itens podem ser verificados em qualquer posição de forma rápida
- Entretanto, o tamanho de vetores é fixo: não podemos adicionar, arbitrariamente, mais elementos
- Um vetor dinâmico é uma estrutura de dados abstrata definida a fim de se gerenciar um vetor internamente, podendo **redimensioná-lo** caso necessário
  - Para tal, um vetor dinâmico deve lembrar do endereço do vetor interno, do seu tamanho, e sua **capacidade**
- Além da vantagem de acesso randômico, é possível para um vetor dinâmico fornecer ao usuário do código o endereço para o **armazenamento interno**
  - Entretanto, **remover itens** de um vetor, exceto que na última posição, é **muito custoso**!

Vetores Dinâmicos

- Vetores são áreas contínuas de memória, cujo principal objetivo é o acesso randômico: itens podem ser verificados em qualquer posição de forma rápida
- Entretanto, o tamanho de vetores é fixo: não podemos adicionar, arbitrariamente, mais elementos
- Um vetor dinâmico é uma estrutura de dados abstrata definida a fim de se gerenciar um vetor internamente, podendo **redimensioná-lo** caso necessário
  - Para tal, um vetor dinâmico deve lembrar do endereço do vetor interno, do seu tamanho, e sua **capacidade**
- Além da vantagem de acesso randômico, é possível para um vetor dinâmico fornecer ao usuário do código o endereço para o **armazenamento interno**
  - Entretanto, **remover itens** de um vetor, exceto que na última posição, é **muito custoso**!

Vetores Dinâmicos

- Vetores são áreas contínuas de memória, cujo principal objetivo é o acesso randômico: itens podem ser verificados em qualquer posição de forma rápida
- Entretanto, o tamanho de vetores é fixo: não podemos adicionar, arbitrariamente, mais elementos
- Um vetor dinâmico é uma estrutura de dados abstrata definida a fim de se gerenciar um vetor internamente, podendo **redimensioná-lo** caso necessário
  - Para tal, um vetor dinâmico deve lembrar do endereço do vetor interno, do seu tamanho, e sua **capacidade**
- Além da vantagem de acesso randômico, é possível para um vetor dinâmico fornecer ao usuário do código o endereço para o **armazenamento interno**
  - Entretanto, **remover itens** de um vetor, exceto que na última posição, é **muito custoso**!

Buffers circulares

- Também chamados de listas circulares, possuem uma implementação similar à de vetores dinâmicos
- A ideia é que, além de salvar o seu tamanho (quantidade de itens), seja lembrada a **posição inicial**
- O vetor interno deve ser imaginado como um círculo: após passarmos da posição final, **voltamos à inicial**
- Graças a isso, podemos salvar itens no começo e no final do *buffer*, similar a uma lista duplamente linkada
- Caso a posição inicial e final sejam iguais, sabemos que o *buffer* está vazio

Buffers circulares

- Também chamados de listas circulares, possuem uma implementação similar à de vetores dinâmicos
- A ideia é que, além de salvar o seu tamanho (quantidade de itens), seja lembrada a **posição inicial**
- O vetor interno deve ser imaginado como um círculo: após passarmos da posição final, **voltamos à inicial**
- Graças a isso, podemos salvar itens no começo e no final do *buffer*, similar a uma lista duplamente linkada
- Caso a posição inicial e final sejam iguais, sabemos que o *buffer* está vazio

Buffers circulares

- Também chamados de listas circulares, possuem uma implementação similar à de vetores dinâmicos
- A ideia é que, além de salvar o seu tamanho (quantidade de itens), seja lembrada a **posição inicial**
- O vetor interno deve ser imaginado como um círculo: após passarmos da posição final, **voltamos à inicial**
- Graças a isso, podemos salvar itens no começo e no final do *buffer*, similar a uma lista duplamente linkada
- Caso a posição inicial e final sejam iguais, sabemos que o *buffer* está vazio

Buffers circulares

- Também chamados de listas circulares, possuem uma implementação similar à de vetores dinâmicos
- A ideia é que, além de salvar o seu tamanho (quantidade de itens), seja lembrada a **posição inicial**
- O vetor interno deve ser imaginado como um círculo: após passarmos da posição final, **voltamos à inicial**
- Graças a isso, podemos salvar itens no começo e no final do *buffer*, similar a uma lista duplamente linkada
- Caso a posição inicial e final sejam iguais, sabemos que o *buffer* está vazio

Buffers circulares

- Também chamados de listas circulares, possuem uma implementação similar à de vetores dinâmicos
- A ideia é que, além de salvar o seu tamanho (quantidade de itens), seja lembrada a **posição inicial**
- O vetor interno deve ser imaginado como um círculo: após passarmos da posição final, **voltamos à inicial**
- Graças a isso, podemos salvar itens no começo e no final do *buffer*, similar a uma lista duplamente linkada
- Caso a posição inicial e final sejam iguais, sabemos que o *buffer* está vazio

Buffers circulares