

Material de Apoio
Aula 5

Manipulação de arrays

Arrays consistem em estruturas de dados capazes de armazenar uma coleção de dados de um mesmo tipo. Maiores informações sobre arrays em:

Exercícios:

1. Implemente a classe `ManipulaArray` definida no Exercício 2 do material: <http://www.inf.unisinos.br/~gersonc/rep/aula12.pdf>.
2. Na classe `ManipulaArray` implemente ainda os seguintes métodos:

- `shiftRight` – onde todos elementos são “empurrados” uma posição para a direita. O primeiro valor passa a ser 0 e o último valor é descartado.

Exemplo:

Antes:	3	4	5	6	7	8	9
Depois:	0	3	4	5	6	7	8

- `shiftLeft` – onde todos os elementos são “empurrados” uma posição para a direita. O primeiro valor é descartado e o último passa a ser 0.

Exemplo:

Antes:	3	4	5	6	7	8	9
Depois:	4	5	6	7	8	9	0

- `insertAt(posição, valor)` – insere o valor `valor` na posição `posição`. Todos elementos após a posição `posição` são “empurrados” uma posição para a direita. O último valor é descartado.

Exemplo: `insertAt(4, 313);`

Antes:	3	4	5	6	7	8	9
Depois:	3	4	5	6	313	7	8

- `removeAt(posição)` – remove o elemento na posição `posição`. Todos elementos após a posição `posição` são “empurrados” uma posição para a esquerda. O último valor passa a ser 0.

Exemplo: `removeAt(4);`

Antes:	3	4	5	6	7	8	9
Depois:	3	4	5	6	8	9	0

Listas lineares com alocação seqüencial

Listas consistem em estruturas de dados capazes de armazenar informações. Objetos “lista” podem ser considerados *containers* de outros objetos ou informações. Uma lista simples pode ser construída a partir de um array, conforme a Figura 1.

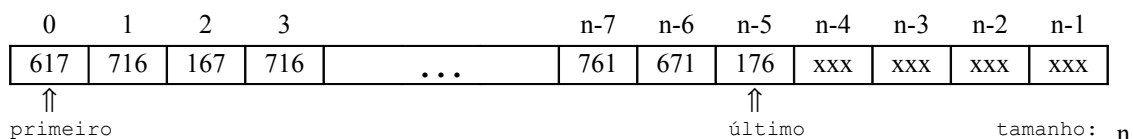


Figura 1. Representação de uma lista com alocação seqüencial.

Na Figura 1 é representado, dentro de um array de `n` posições, uma lista cuja representação de memória foi alocada previamente. Em outras palavras, o tamanho máximo da lista é o tamanho do array. Nesta figura também é dado destaque a três informações mínimas necessárias para manipulação de uma lista seqüencial dentro de um array: além da informação sobre o tamanho do array, é importante também ter referências para a primeira e a última posição no array ocupada pela lista. Nesta representação o tamanho da lista é obtido pela simples subtração dos valores dos ponteiros “primeiro” e “último”.

Atenção: um cuidado a tomar em Java é que os arrays iniciam sua indexação em 0. Assim, deve ser tomado um cuidado especial na manipulação dos ponteiros primeiro e último. Duas alternativas para implementação são dadas. Na primeira, quando os ponteiros `primeiro` e `último` forem iguais, significa que a lista contém apenas um elemento. Neste caso, aconselha-se atribuir um valor negativo para estes ponteiros quando a lista estiver vazia. Outra alternativa é fazer com que o ponteiro `último` aponte para a próxima posição do vetor que estiver vaga para escrita: desta forma, quando `primeiro` for igual a `último`, a lista estará vazia. Ainda neste caso, quando `último` for igual ao tamanho do array, a lista estará cheia.

As operações básicas a serem suportadas em uma lista são inserção de um novo elemento e retirada de um elemento da lista. Diversas políticas de inserção e remoção podem ser empregadas.

Políticas de inserção:

- `push(elemento)` – insere o elemento no início da lista, o que força o deslocamento de todos elementos da lista em uma posição.
- `push_back(elemento)` – insere elemento no fim da lista;
- `insertAt(posição, elemento)` – insere elemento na posição `posição`, deslocando o conteúdo da lista a partir desta posição da forma que o elemento na posição `i` passe a ocupar a posição `i+1`.

Políticas de remoção:

- `pop` – Retira elemento do início da lista, retorna o valor, e atualiza a lista realizando uma das seguintes alternativas:
 - faz com que `primeiro` seja igual a `primeiro + 1`, fazendo com que a posição onde estava o primeiro elemento fique desprezada. O ponteiro `último` não necessita ser atualizado.
 - desloca todos elementos da lista, passando o elemento da posição `i` para a posição `i-1`;
- `pop_back` – retira um elemento do fim da lista, retorna o valor e atualiza o ponteiro `último` para a posição imediatamente anterior a que estava.
- `getAt(posição)` – retira elemento na posição `posição`, deslocando o conteúdo da lista a partir desta posição passando o elemento da posição `i` para a posição `i-1`.

Com estas operações, duas situações de erro podem ocorrer:

- *Overflow*: ocorre quando a lista já ocupa todo o array e o usuário tenta inserir um novo elemento.
- *Underflow*: ocorre quando a lista está vazia e o usuário tenta retirar um elemento.

Ainda útil é adicionar a lista a possibilidade de realizar operações de acesso:

- `begin` – retorna o primeiro elemento da lista;
- `end` – retorna o último elemento da lista;
- `read(posição)` – retorna o elemento da lista que está na posição `posição`;
- `find(elemento)` – retorna a posição que o elemento se encontra na lista;
- `length` – retorna o tamanho da lista (quantos elementos ela possui).

Exercícios:

1. Implemente uma classe `ListaA`, uma classe dedicada a manutenção de uma lista de números inteiros. Esta classe deve possuir no seu estado interno uma array de inteiros e permitir sua manipulação com as políticas `push` e `pop`. Também deve permitir todas as operações de acesso definidas previamente. Esta classe deve ter, pelo menos, um construtor que recebe o tamanho máximo da lista (o tamanho do array).
2. Implemente uma classe `ListaB`, uma classe dedicada a manutenção de uma lista de números inteiros. Esta classe deve possuir no seu estado interno uma array de inteiros e permitir sua manipulação com as políticas `push_back` e `pop_back`. Também deve permitir todas as operações de acesso definidas previamente. Esta classe deve ter, pelo menos, um construtor que recebe o tamanho máximo da lista (o tamanho do array).
3. Implemente uma classe `ListaC`, uma classe dedicada a manutenção de uma lista de números inteiros. Esta classe deve possuir no seu estado interno uma array de inteiros e permitir sua manipulação com as políticas, `push`, `pop`, `push_back`, `pop_back`, `insertAt` e `getAt`. Também deve permitir todas as operações de acesso definidas previamente. Esta classe deve ter, pelo menos, um construtor que recebe o tamanho máximo da lista (o tamanho do array).
4. Implemente suporte com tratamento de exceções para a ocorrência de situações de *overflow* e *underflow* a classe desenvolvida no Exercício 3.
5. Quando pode ser vantajosa a implementação que, quando retirado um elemento do início da lista, o ponteiro `primeiro` passa a apontar para a próxima posição válida, ficando a antiga posição anterior desocupada na lista (mas ainda ocupando memória do array)?
6. A situação apontada no exercício anterior pode ocasionar uma situação indesejada de *overflow*, quando for necessário inserir um novo elemento no fim da lista e o ponteiro `último` já indicar plena ocupação do array e existir espaço livre nas posições anteriores ao ponteiro `primeiro`. O que pode ser feito para contornar esta situação?
7. Reimplemente a classe desenvolvida no Exercício 3 para armazenar objetos `String`.