Universidade de Brasília

Filas e Pilhas

Algoritmos e Estruturas de Dados

Prof. Daniel Guerreiro e Silva

Roteiro

1. Pilhas
2. Filas

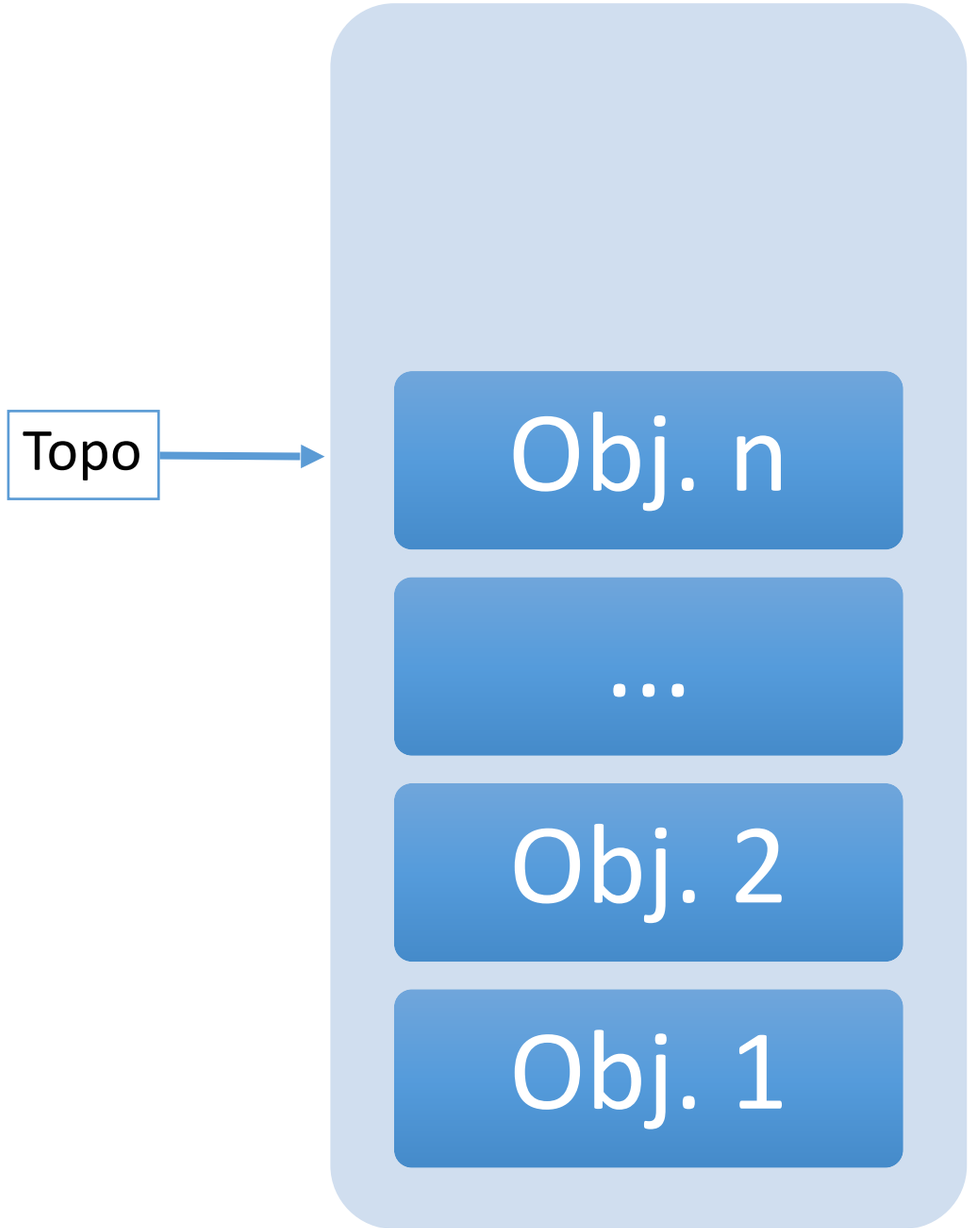
Leitura: Seções 4.1, 4.2, 4.4, 4.5 do livro-texto (Drozdek).

Pilhas

- Assim como a lista ligada, a pilha é uma estrutura de dados linear
- Ela tem uma particularidade: o elemento removido do conjunto é especificado previamente, i.e. é **sempre o mais recentemente inserido**
- Pilhas implementam a política conhecida como **LIFO (last-in, first-out)**
- A ordem de retirada de elementos de uma pilha é o **inverso** da ordem de inserção

Pilhas

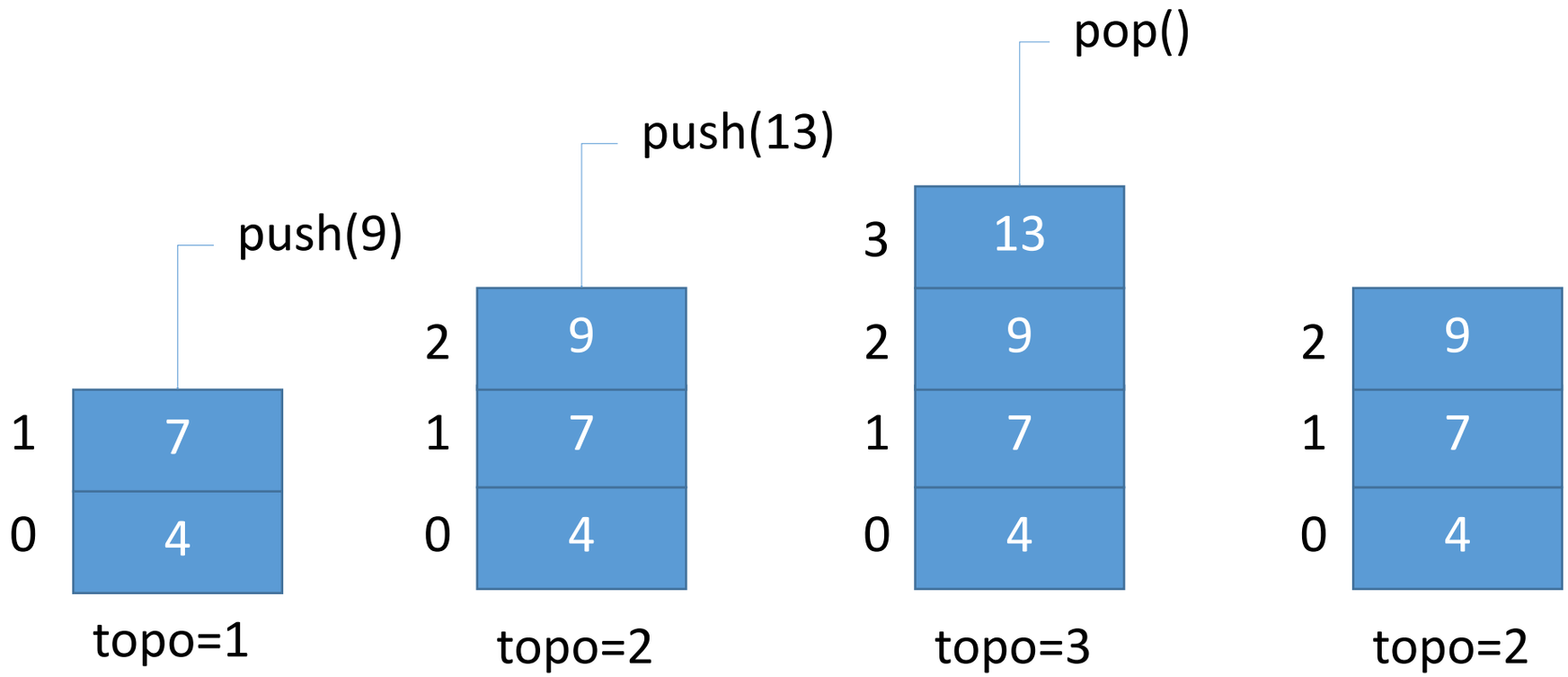
- Pilhas são facilmente implementadas com arranjos lineares, por exemplo vetores ou listas ligadas
- É fundamental ter na implementação a posição do topo da pilha, i.e. a posição onde foi inserido o elemento mais recente



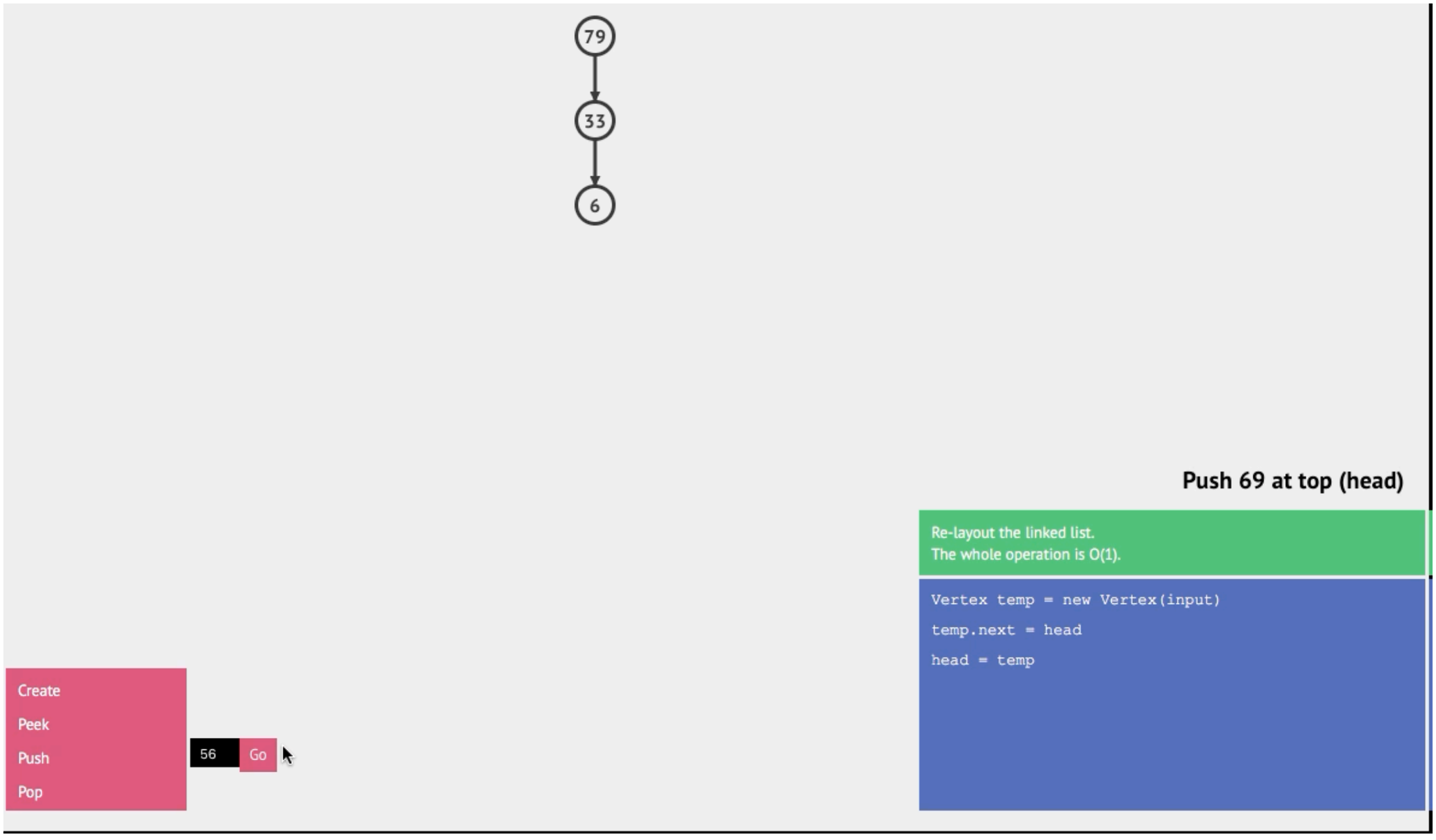
Operações em uma pilha

- empty()
  - Testa se a pilha está vazia
- push(elemento)
  - Insere elemento no topo da pilha
- pop()
  - Retira um elemento do topo da pilha

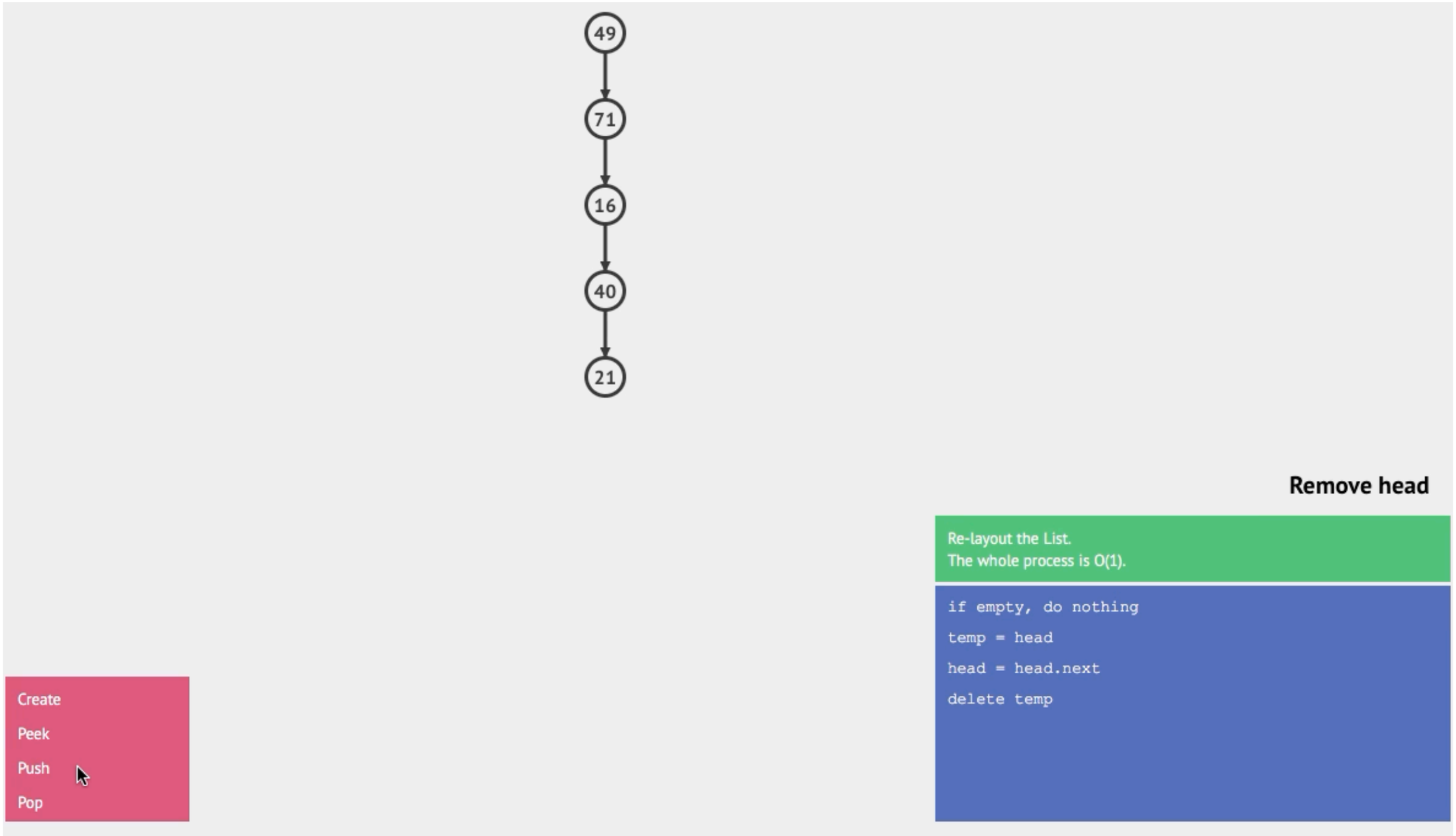
Operações em uma pilha



Push


79
33
6
Push 69 at top (head)
Re-layout the linked list.
The whole operation is O(1).
Vertex temp = new Vertex(input)
temp.next = head
head = temp
Create
Peek
Push
Pop
56
Go

Pop


49
71
16
40
21
Remove head
Re-layout the List.
The whole process is O(1).
if empty, do nothing
temp = head
head = head.next
delete temp
Create
Peek
Push
Pop

Implementação em C++

- Podemos implementar usando array, vector ou uma implementação qualquer de lista ligada
- As operações de push / pop, independentemente da implementação têm custo computacional $O(1)$
- Porém, com vector/array, ao encher a pilha, a realocação de memória gera uma cópia dos dados para a nova área com custo $O(n)$

Pilha de double usando arrays – doubleStack.h

```
class doubleStack{
public:
  doubleStack(int max_elements = 1024);
  ~doubleStack();

  double pop(); //remove e retorna topo da pilha
  void push(double); //insere no topo da pilha
  double peek(); //somente consulta valor do topo
  bool isEmpty(); //true se pilha estiver vazia

private:
  double *data;
  int topo;
};
```

Exercício guiado: vamos implementar os métodos declarados ao lado.

Solução

- doubleStack.cpp (no Aprender 3)
- E se a pilha chega ao limite de armazenamento?
  - Não armazena mais nada, ou
  - Rotina de realocação da pilha em nova área de memória maior -> cópia dos n elementos

Pilha de objetos de tipo genérico usando estrutura list da STL

Formatos ou Templates é um recurso do C++ que permite escrever funções e classes que manipulam tipos genéricos de dados.

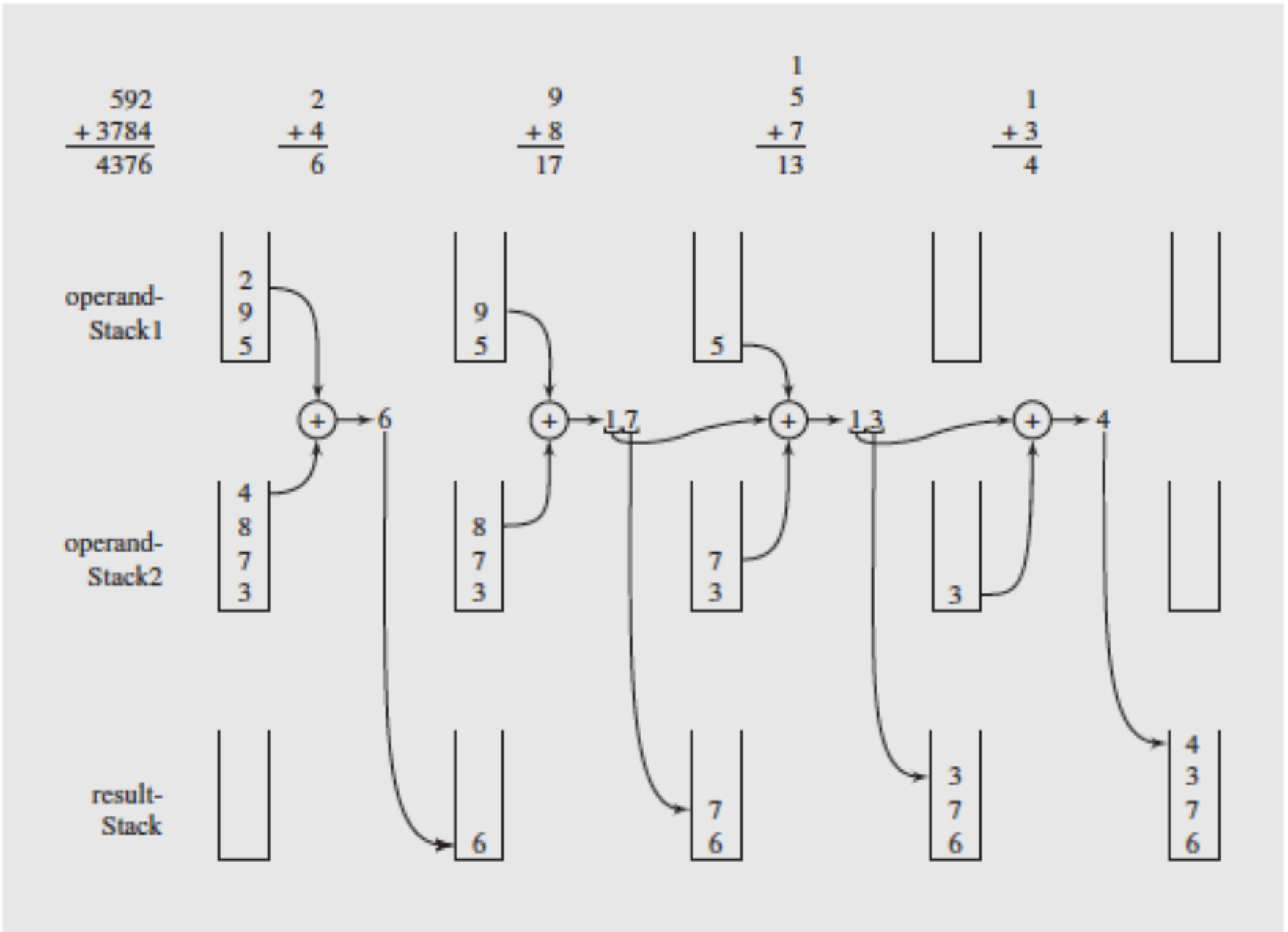
- Para declarar um tipo genérico denominado T use o comando

```
template<class T>
```

- Veja implementação em genListStack.h

Exercício de Assimilação de Conceitos

Crie um programa que utiliza pilhas para somar números muito grandes

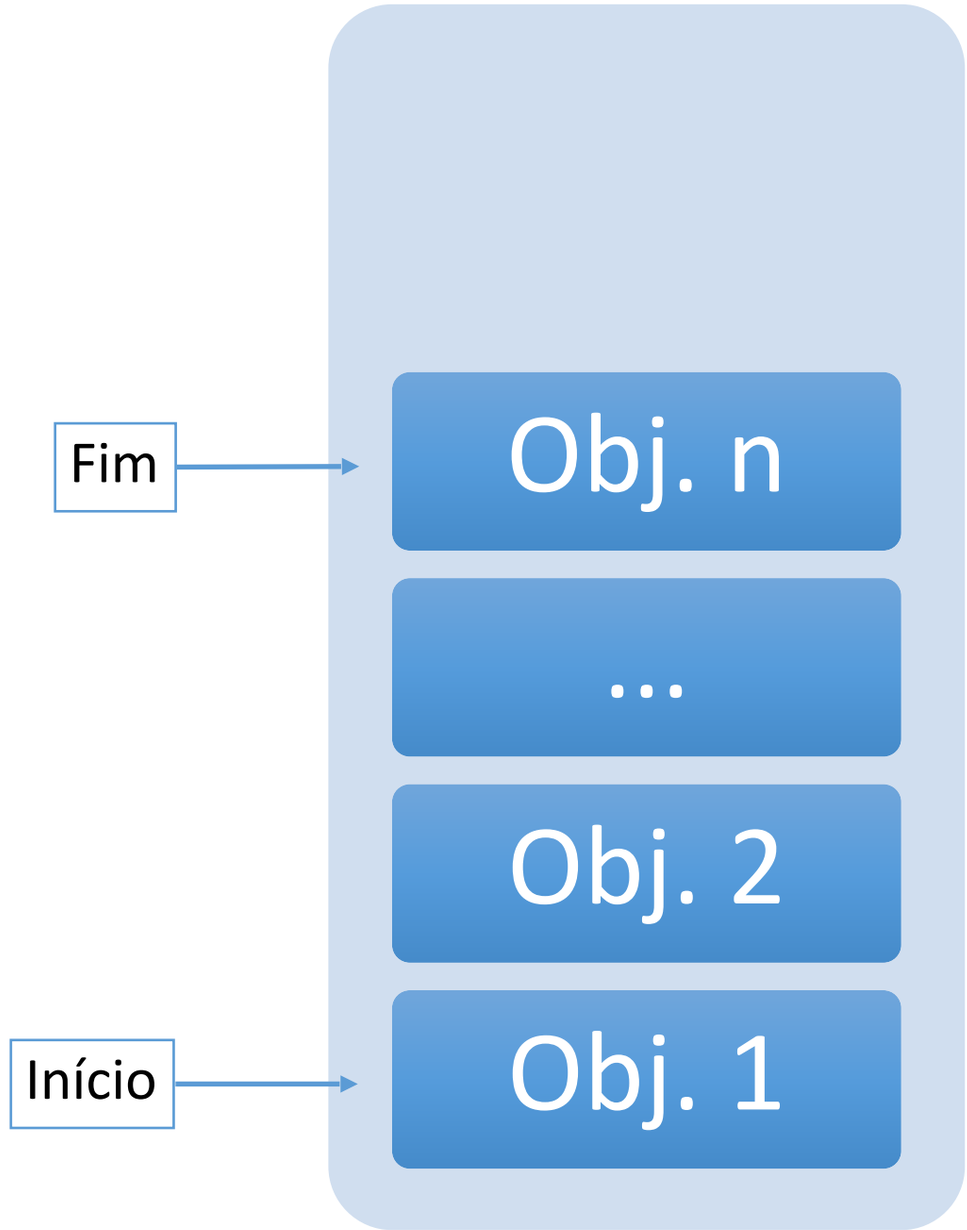


Filas

- A fila é uma estrutura de dados linear que cresce inserindo elementos no final e diminui removendo elementos no seu início
- Filas implementam a política conhecida como **FIFO (first-in, first-out)**
- A ordem de retirada de elementos de uma pilha é **igual** à ordem de inserção

Filas

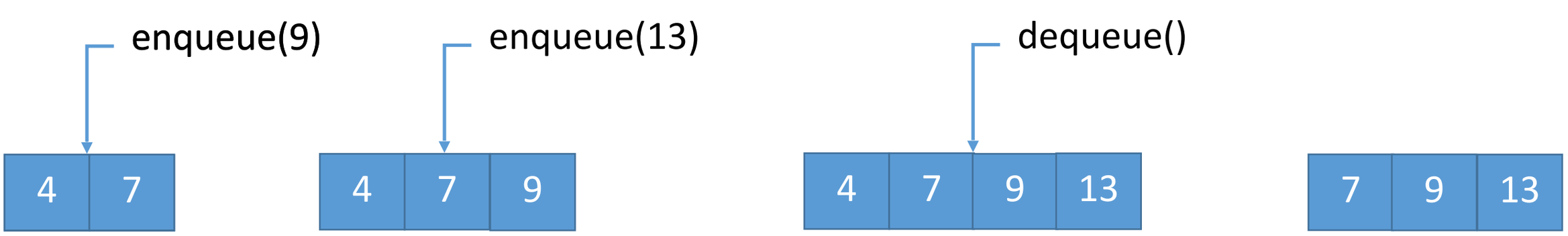
- Filas também são facilmente implementadas com arranjos lineares, por exemplo vetores ou listas ligadas
- É necessário ter na implementação a posição do início e do final da fila



Operações em uma fila

- isEmpty()
  - Testa se a fila está vazia
- enqueue (elemento)
  - Insere elemento no final da fila
- dequeue()
  - Retira um elemento do início da fila
- front()
  - Retorna o primeiro da fila, sem removê-lo

Operações em uma fila



Implementação de filas em C++

- Podemos implementar usando array, vector ou uma implementação qualquer de lista ligada
- Porém, usar uma lista duplamente encadeada é a melhor escolha, visto que as operações de retirar do início da fila e colocar no final da fila terão custo $O(1)$

Fila de objetos de tipo genérico usando estrutura list da STL

```
template<class T>
class Queue {
public:
  Queue();
  void clear();
  bool isEmpty();
  T& front();
  T dequeue();
  void enqueue(const T& el);
private:
  list<T> lst;
};
```

- Veja o restante da implementação no arquivo genQueue.h

Pilhas na STL

- Pilhas usam o adaptador de contêiner `stack`
  - Usa um contêiner (`deque` é o padrão) para se comportar de um modo específico
- Declaração:

```
stack<tipo> variavel;
```

`bool empty()`	Retorna true se a pilha está vazia
`size_type size()`	Retorna o tamanho da pilha
`void push (const value_type& val)`	Insere elemento no topo
`void pop()`	Apaga elemento do topo
`value_type& top()`	Retorna elemento do topo

Filas na STL

- Filas usam o adaptador de contêiner `queue`, que por padrão é implementado usando o contêiner `deque`.
- Declaração:

```
queue<tipo> variavel;
```

`value_type& front()`	Retorna o primeiro elemento da fila
`value_type& back()`	Retorna o último elemento da fila
`void push (const value_type& val)`	Insere elemento no final da fila
`void pop()`	Remove o primeiro elemento da fila
`size_type size()`	Retorna o tamanho da fila