BCC702 - Programação de Computadores II

Adaptadores de Containers
Algoritmos da STL

Profª Valéria de Carvalho Santos

Slides adaptados do profº Alan de Freitas

Adaptadores de Container

- A STL fornece três adaptadores de container: *stack*, *queue* e *priority_queue*
- Não são containers de primeira classe:
  - Não fornecem a implementação real da estrutura
  - Não suportam iteradores
- As três classes fornecem as funções-membro *push* e *pop* para inserir e remover elementos, respectivamente.

Adaptador stack

- Pilha
  - LIFO = *last-in*, *first-out*
  - permite inserções e remoções apenas em uma extremidade da estrutura
  - pode ser implementada com qualquer um dos containers de sequência: *vector*, *deque* ou *list* (por padrão é implementada com deque)

Adaptador stack

- Principais funções-membro
  - *push*: insere elemento no topo
  - *pop*: remove elemento do topo
  - *top*: retorna o elemento do topo
  - *size*: retorna a quantidade de elementos
  - *empty*: verifica se a pilha está vazia

Adaptador stack

- Declaração

  stack**<tipo>** nome;

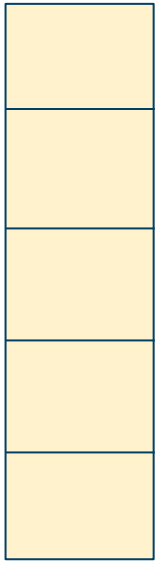
  stack**<tipo, containerSequencial<tipo>>** nome;

Adaptador stack

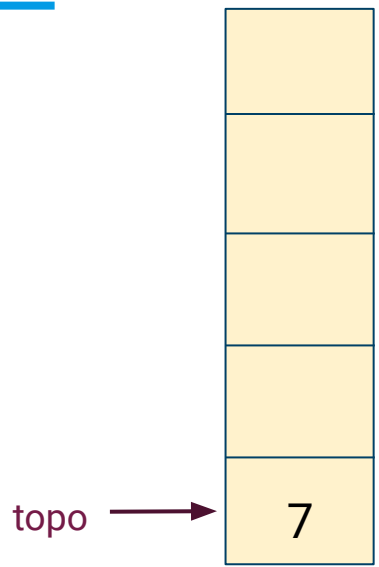
- Declaração

stack<**tipo**> nome;
stack<**string**> pilhaNomes;

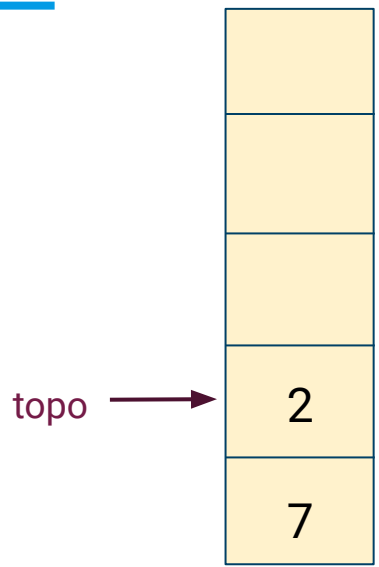
stack<**tipo, containerSequencial<tipo>>** nome;
stack<**string**, **list<string>**> pilhaProdutos;



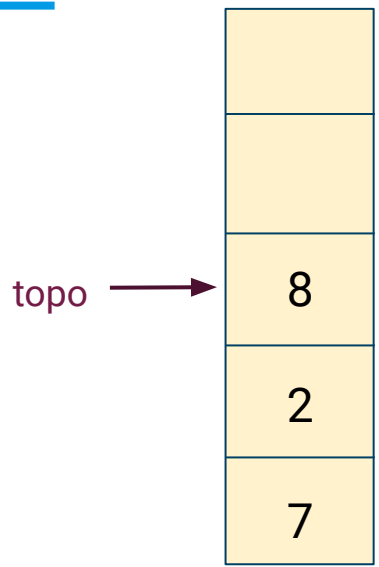
```cpp
#include <iostream>
#include <stack>
using namespace std;

int main(){
    stack<int> pilha;









}
```



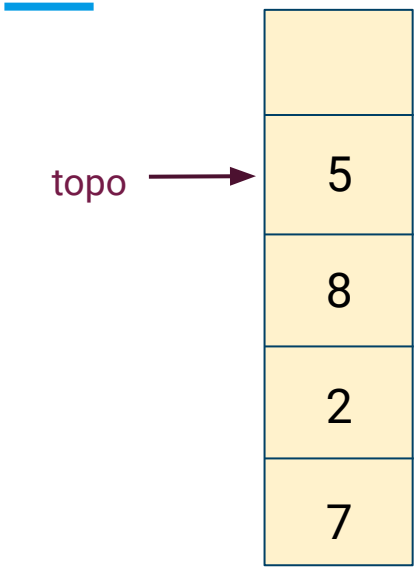

```
#include <iostream>
#include <stack>
using namespace std;

int main(){
    stack<int> pilha;
    pilha.push(7);




}
```



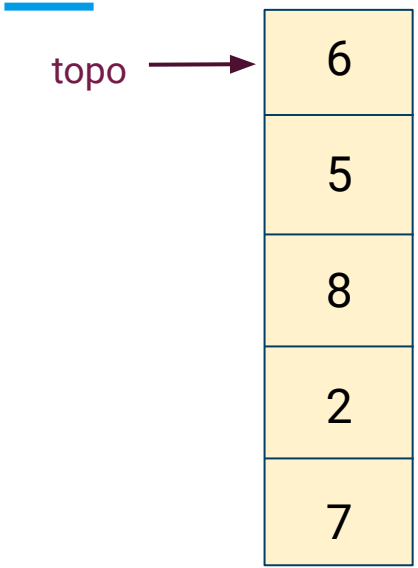

```
#include <iostream>
#include <stack>
using namespace std;

int main(){
    stack<int> pilha;
    pilha.push(7);
    pilha.push(2);









}
```



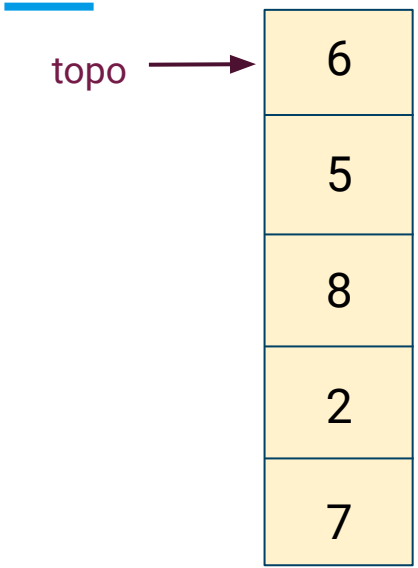

```
#include <iostream>
#include <stack>
using namespace std;

int main(){
    stack<int> pilha;
    pilha.push(7);
    pilha.push(2);
    pilha.push(8);









}
```




```
#include <iostream>
#include <stack>
using namespace std;

int main(){
    stack<int> pilha;
    pilha.push(7);
    pilha.push(2);
    pilha.push(8);
    pilha.push(5);




}
```

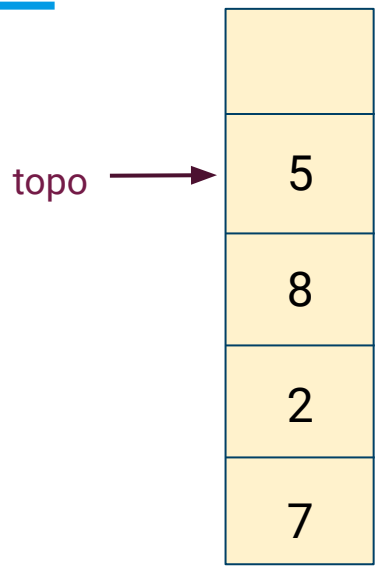



```
#include <iostream>
#include <stack>
using namespace std;

int main(){
    stack<int> pilha;
    pilha.push(7);
    pilha.push(2);
    pilha.push(8);
    pilha.push(5);
    pilha.push(6);




}
```



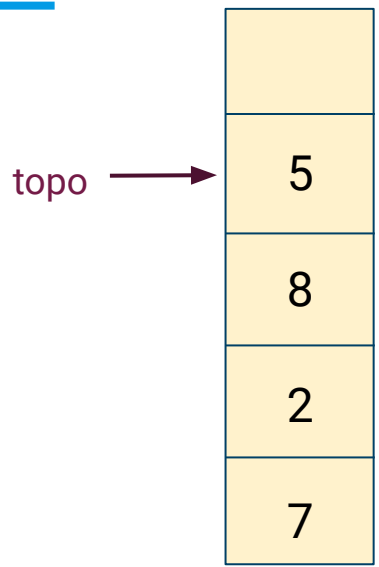

```
#include <iostream>
#include <stack>
using namespace std;

int main(){
    stack<int> pilha;
    pilha.push(7);
    pilha.push(2);
    pilha.push(8);
    pilha.push(5);
    pilha.push(6);
    cout << "Topo: " << pilha.top() << endl;



}
```

```
Topo: 6
```



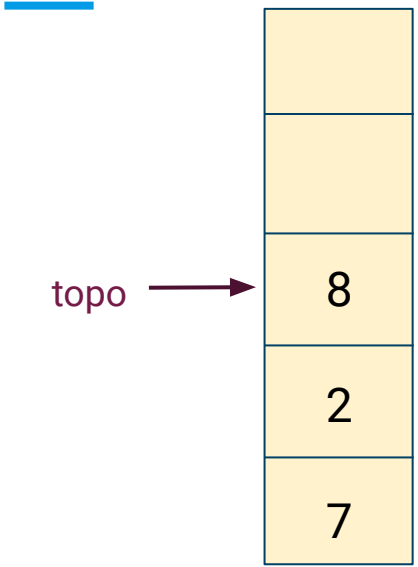

```
#include <iostream>
#include <stack>
using namespace std;

int main(){
    stack<int> pilha;
    pilha.push(7);
    pilha.push(2);
    pilha.push(8);
    pilha.push(5);
    pilha.push(6);
    cout << "Topo: " << pilha.top() << endl;
    pilha.pop();



}
```

```
Topo: 6
```




```
#include <iostream>
#include <stack>
using namespace std;

int main(){
    stack<int> pilha;
    pilha.push(7);
    pilha.push(2);
    pilha.push(8);
    pilha.push(5);
    pilha.push(6);
    cout << "Topo: " << pilha.top() << endl;
    pilha.pop();
    cout << "Topo: " << pilha.top() << endl;


}
```

```
Topo: 6
Topo: 5
```




```
#include <iostream>
#include <stack>
using namespace std;

int main(){
    stack<int> pilha;
    pilha.push(7);
    pilha.push(2);
    pilha.push(8);
    pilha.push(5);
    pilha.push(6);
    cout << "Topo: " << pilha.top() << endl;
    pilha.pop();
    cout << "Topo: " << pilha.top() << endl;
    pilha.pop();
    cout << "Topo: " << pilha.top() << endl;
}
```

```
Topo: 6
Topo: 5
Topo: 8
```

Exercício

- Faça uma função que recebe uma pilha (stack) por parâmetro e esvazia a pilha.

Adaptador queue

- Fila
  - FIFO: *first-in*, *first-out*
  - Permite inserções apenas no final da estrutura e remoções apenas no início
  - pode ser implementada com *list* ou *deque* (por padrão, deque)

Adaptador queue

- Principais funções-membro:
  - *push*: insere um elemento no final
  - *pop*: remove um elemento da frente
  - *front*: retorna o elemento da frente
  - *back*: retorna o elemento de trás
  - *empty*: verifica se a fila está vazia
  - *size*: retorna o número de elementos da fila

Adaptador stack

- Declaração

queue**<tipo>** nome;

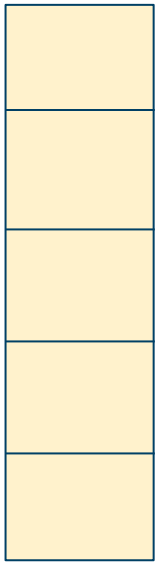
queue**<tipo, containerSequencial<tipo>>** nome;

Adaptador stack

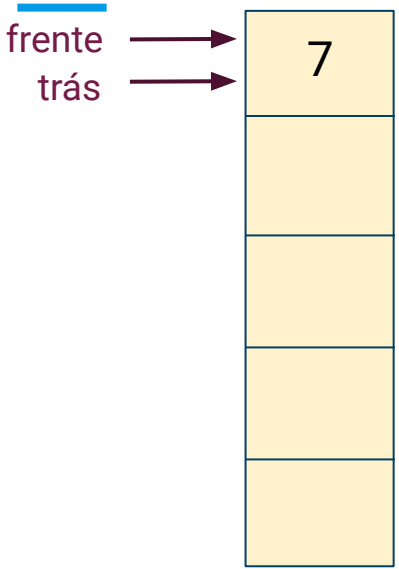
- Declaração

queue**<tipo>** nome;
queue<**string**> pilhaNomes;

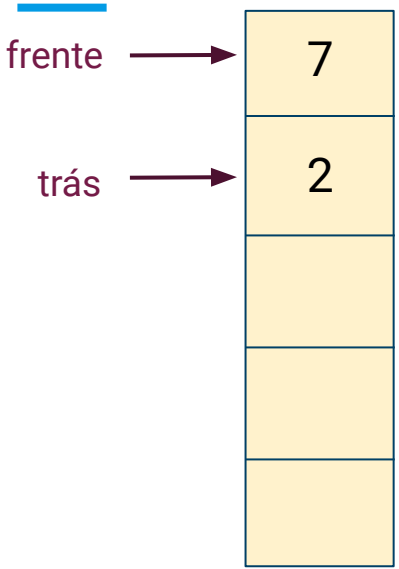
queue**<tipo, containerSequencial<tipo>>** nome;
queue<**string**, **list<string>**> pilhaProdutos;



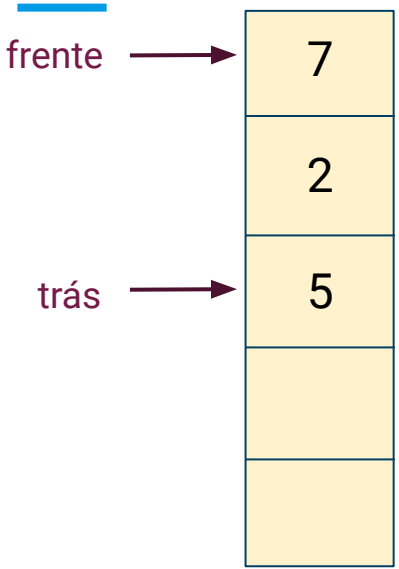
```cpp
#include <iostream>
#include <queue>
using namespace std;

int main(){
    queue<int> fila;









}
```



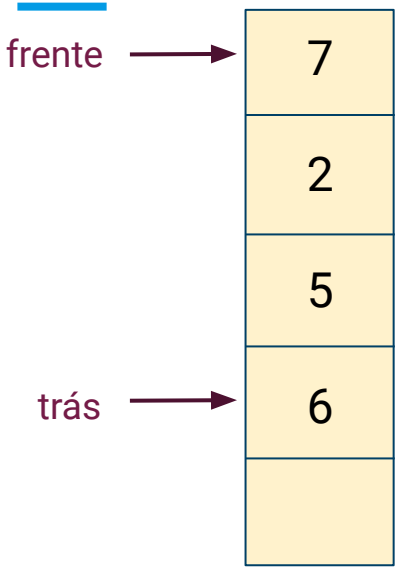

```
#include <iostream>
#include <queue>
using namespace std;

int main(){
    queue<int> fila;
    fila.push(7);









}
```



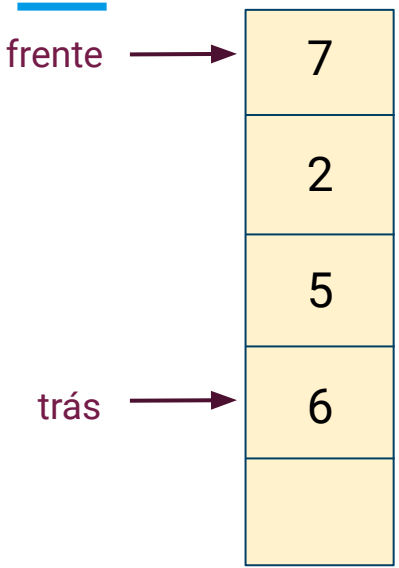

```
#include <iostream>
#include <queue>
using namespace std;

int main(){
    queue<int> fila;
    fila.push(7);
    fila.push(2);









}
```




```
#include <iostream>
#include <queue>
using namespace std;

int main(){
    queue<int> fila;
    fila.push(7);
    fila.push(2);
    fila.push(5);









}
```

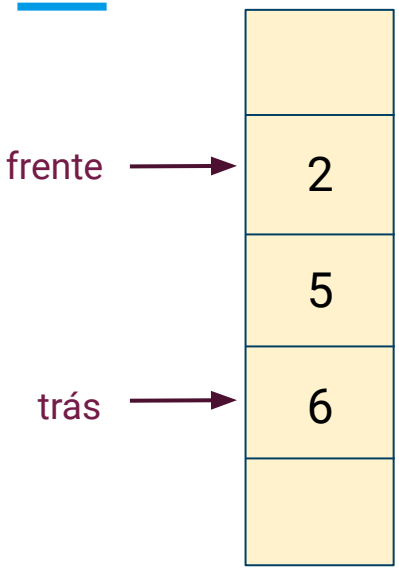



```
#include <iostream>
#include <queue>
using namespace std;

int main(){
    queue<int> fila;
    fila.push(7);
    fila.push(2);
    fila.push(5);
    fila.push(6);









}
```



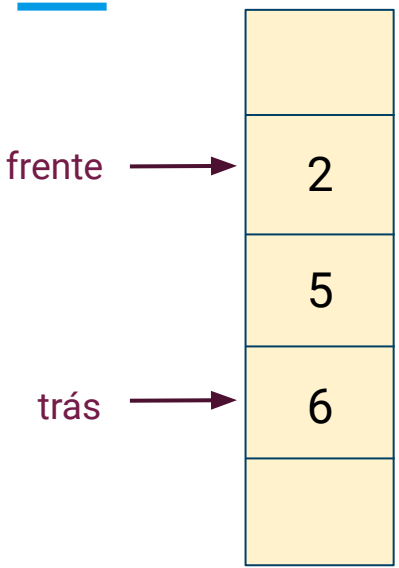

```
#include <iostream>
#include <queue>
using namespace std;

int main(){
    queue<int> fila;
    fila.push(7);
    fila.push(2);
    fila.push(5);
    fila.push(6);
    cout << "Frente: " << fila.front() << endl;
    cout << "Trás: " << fila.back() << "\n" << endl;









}
```

```
Frente: 7
Trás: 6
```



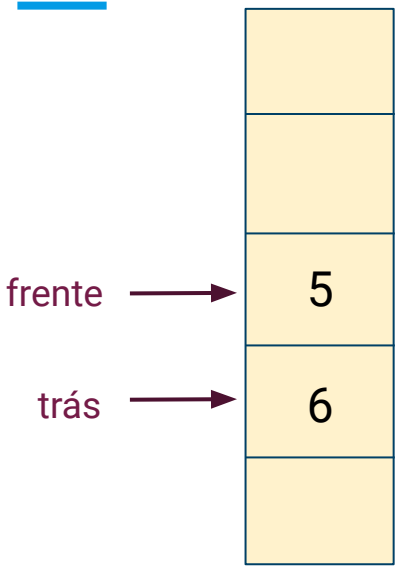

```
#include <iostream>
#include <queue>
using namespace std;

int main(){
    queue<int> fila;
    fila.push(7);
    fila.push(2);
    fila.push(5);
    fila.push(6);
    cout << "Frente: " << fila.front() << endl;
    cout << "Trás: " << fila.back() << "\n" << endl;
    fila.pop();









}
```

```
Frente: 7
Trás: 6
```



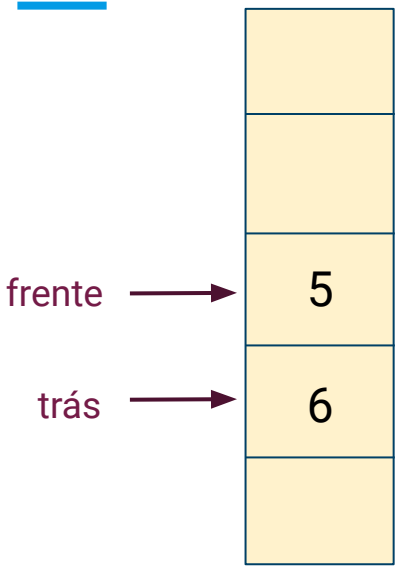

```
#include <iostream>
#include <queue>
using namespace std;

int main(){
    queue<int> fila;
    fila.push(7);
    fila.push(2);
    fila.push(5);
    fila.push(6);
    cout << "Frente: " << fila.front() << endl;
    cout << "Trás: " << fila.back() << "\n" << endl;
    fila.pop();
    cout << "Frente: " << fila.front() << endl;
    cout << "Final: " << fila.back() << "\n" << endl;




}
```

```
Frente: 7
Trás: 6

Frente: 2
Trás: 6
```



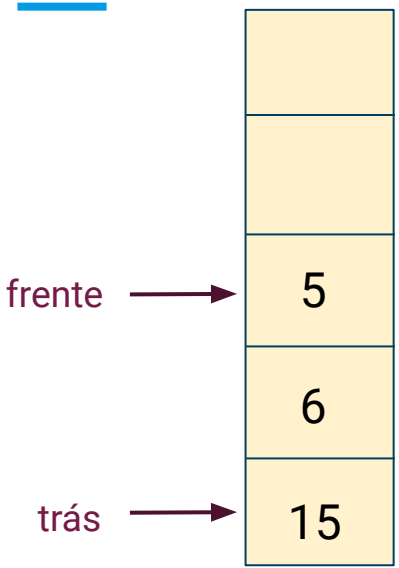

```
#include <iostream>
#include <queue>
using namespace std;

int main(){
    queue<int> fila;
    fila.push(7);
    fila.push(2);
    fila.push(5);
    fila.push(6);
    cout << "Frente: " << fila.front() << endl;
    cout << "Trás: " << fila.back() << "\n" << endl;
    fila.pop();
    cout << "Frente: " << fila.front() << endl;
    cout << "Final: " << fila.back() << "\n" << endl;
    fila.pop();




}
```

```
Frente: 7
Trás: 6

Frente: 2
Trás: 6
```



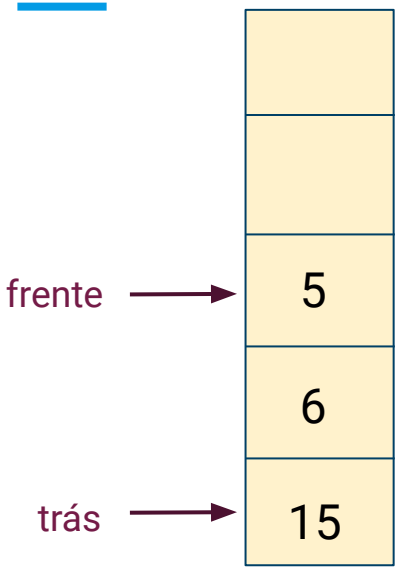

```
#include <iostream>
#include <queue>
using namespace std;

int main(){
    queue<int> fila;
    fila.push(7);
    fila.push(2);
    fila.push(5);
    fila.push(6);
    cout << "Frente: " << fila.front() << endl;
    cout << "Trás: " << fila.back() << "\n" << endl;
    fila.pop();
    cout << "Frente: " << fila.front() << endl;
    cout << "Final: " << fila.back() << "\n" << endl;
    fila.pop();
    cout << "Frente: " << fila.front() << endl;
    cout << "Final: " << fila.back() << "\n" << endl;



}
```

```
Frente: 7
Trás: 6

Frente: 2
Trás: 6

Frente: 5
Trás: 6
```




```
#include <iostream>
#include <queue>
using namespace std;

int main(){
    queue<int> fila;
    fila.push(7);
    fila.push(2);
    fila.push(5);
    fila.push(6);
    cout << "Frente: " << fila.front() << endl;
    cout << "Trás: " << fila.back() << "\n" << endl;
    fila.pop();
    cout << "Frente: " << fila.front() << endl;
    cout << "Final: " << fila.back() << "\n" << endl;
    fila.pop();
    cout << "Frente: " << fila.front() << endl;
    cout << "Final: " << fila.back() << "\n" << endl;
    fila.push(15);


}
```

```
Frente: 7
Trás: 6

Frente: 2
Trás: 6

Frente: 5
Trás: 6
```




```
#include <iostream>
#include <queue>
using namespace std;

int main(){
    queue<int> fila;
    fila.push(7);
    fila.push(2);
    fila.push(5);
    fila.push(6);
    cout << "Frente: " << fila.front() << endl;
    cout << "Trás: " << fila.back() << "\n" << endl;
    fila.pop();
    cout << "Frente: " << fila.front() << endl;
    cout << "Final: " << fila.back() << "\n" << endl;
    fila.pop();
    cout << "Frente: " << fila.front() << endl;
    cout << "Final: " << fila.back() << "\n" << endl;
    fila.push(15);
    cout << "Frente: " << fila.front() << endl;
    cout << "Final: " << fila.back() << endl;
}
```

```
Frente: 7
Trás: 6

Frente: 2
Trás: 6

Frente: 5
Trás: 6

Frente: 5
Trás: 15
```

Adaptador priority_queue

- Implementa uma fila de prioridades
- Os elementos são inseridos na ordem de prioridade, sendo que o elemento de maior prioridade (maior valor) será o primeiro elemento removido
- pode ser implementada com vector ou deque (por padrão, vector)

Adaptador priority_queue

- Principais funções-membro:
  - *push*: insere um elemento com base na prioridade
  - *pop*: remove o elemento de maior prioridade
  - *top*: retorna o elemento de maior prioridade
  - *empty*: verifica se a fila está ou não vazia
  - *size*: retorna o número de elementos da fila

```
#include <iostream>
#include <queue>
using namespace std;

int main(){
    priority_queue<double> fila;
    fila.push(3.2);
    fila.push(5.6);
    fila.push(8.1);
    fila.push(4.9);

    while(!fila.empty()){
        cout << fila.top() << endl;
        fila.pop();
    }
}
```

```cpp
#include <iostream>
#include <queue>
using namespace std;

int main(){
    priority_queue<double> fila;
    fila.push(3.2);
    fila.push(5.6);
    fila.push(8.1);
    fila.push(4.9);

    while(!fila.empty()){
        cout << fila.top() << endl;
        fila.pop();
    }
}
```

```
8.1
5.6
4.9
3.2
```

Algoritmos da STL

- A biblioteca STL disponibiliza algoritmos para executar operações comuns a dados armazenados em containers
  - Exemplos: ordenação, busca, cópia, somatório, etc.
- Normalmente, são implementações eficientes dessas operações
  - poupa tempo de implementação
  - implementação confiável

Algoritmos da STL

- O cabeçalho <algorithm> inclui algoritmos comuns para containers
- O cabeçalho <numeric> inclui algoritmos comuns para dados numéricos
- A lista completa de algoritmos disponíveis está em http://www.cplusplus.com/reference/algorithm/

Exemplo - sort

```
#include <iostream>
#include <algorithm>
using namespace std;

int main(){
    int numeros[] = {32,71,12,45,26,80,53,33};
    vector<int> v(numeros, numeros+8);

    //ordena os 4 primeiros elementos
    sort(v.begin(), v.begin()+4);

    //ordena tudo
    sort(v.begin(), v.end());
}
```

Exemplo - fill e fill_n

```
#include <iostream>
#include <algorithm>
using namespace std;

int main(){
    vector<char> v(10);

    //preenche o container de begin() a end() com B
    //B B B B B B B B B B
    fill(v.begin(), v.end(), 'B');

    //preenche os 5 primeiros elementos com A
    //A A A A A B B B B B
    fill_n(v.begin(), 5, 'A');
}
```

Exemplo - binary_search

```
#include <iostream>
#include <algorithm>
using namespace std;

int main(){
    int numeros[] = {32,71,12,45,26,80,53,33};
    vector<int> v(numeros, numeros+8);

    //ordena tudo
    sort(v.begin(), v.end());

    //verifica se o elemento se encontra no container
    if(binary_search(v.begin(), v.end(), 12)){
        cout << "Elemento encontrado" << endl;
    }
}
```

Exemplo - max_element

```
#include <iostream>
#include <algorithm>
using namespace std;

int main(){
    int numeros[] = {32,71,12,45,26,80,53,33};
    vector<int> v(numeros, numeros+8);

    //ordena tudo
    int r = *max_element(v.begin(), v.end());

    cout << "Maior elemento" << r << endl;

}
```