

# «Stack» e «Queue»

**Kaio Christaldo**  
**Fabricio Matsunaga**

**Template**  **Stack**

# Apresentação Problema Motivador

beecrowd | 1068



## Balanço de Parênteses I

Por Neilor Tonin, URI  Brasil

**Timelimit: 1**

Dada uma expressão qualquer com parênteses, indique se a quantidade de parênteses está correta ou não, sem levar em conta o restante da expressão. Por exemplo:

$a + (b * c) - 2 - a$       está correto

$(a + b * (2 - c) - 2 + a) * 2$       está correto

enquanto

$(a * b - (2 + c)$       está incorreto

$2 * (3 - a) )$       está incorreto

$) 3 + b * (2 - c) ($       está incorreto

Ou seja, todo parênteses que fecha deve ter um outro parênteses que abre correspondente e não pode haver parênteses que fecha sem um previo parenteses que abre e a quantidade total de parenteses que abre e fecha deve ser igual.

### Entrada

Como entrada, haverá  $N$  expressões ( $1 \leq N \leq 10000$ ), cada uma delas com até 1000 caracteres.


### Saída

O arquivo de saída deverá ter a quantidade de linhas correspondente ao arquivo de entrada, cada uma delas contendo as palavras **correct** ou **incorrect** de acordo com as regras acima fornecidas.

**1068 – Balanço de Parênteses I**

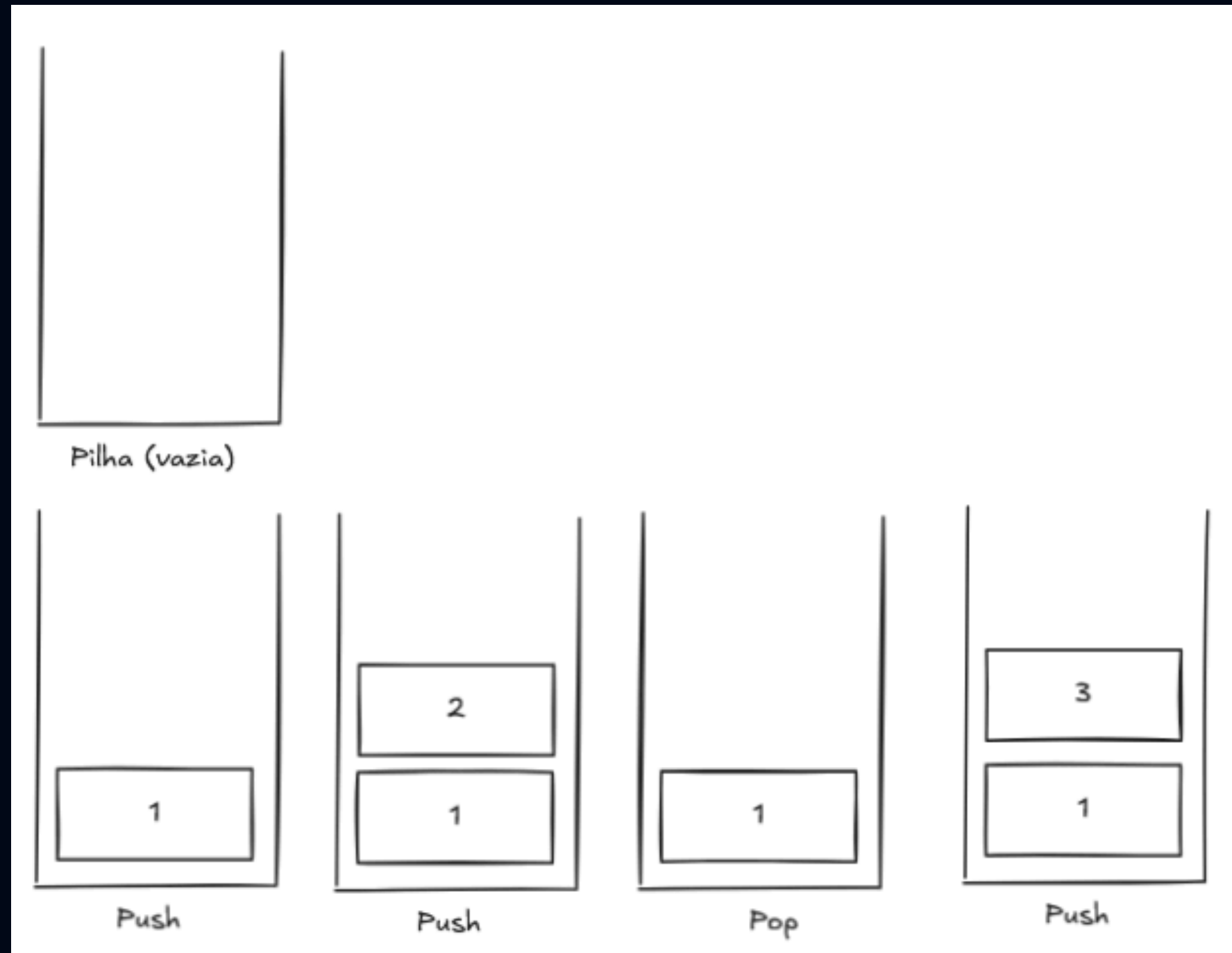
# Template <stack>

- **Definição:** Uma pilha (**stack**) é uma estrutura de dados que oferece duas operações com **tempo  $O(1)$** :
  - Adicionar um elemento no topo.
  - Remover um elemento do topo.
- É possível acessar apenas o elemento que está no **topo da pilha**..
- **Implementação:** Baseada em um **<deque>** ou **<vector>**
- **Cabeçalho necessário:**



```
1  #include <iostream>
2  #include <stack>
```

# Template <Stack>



- **Analogia: Pilha de Pratos**
  - o o ultimo prato empilhado será o primeiro prato retirado da pilha
- **Cuidado:**
  - o Tentar remover em uma pilha.
    - Sempre verifique se a pilha está vazia antes de remoções.
- **Muito Utilizado em:**
  - o Algoritmos de Ordenação
  - o Manipulação de Dados
  - o Grafos

# Template <stack>

## Operações na Stack

### Principais Operações Usadas:

- **s.push(value)** – **Insere** um elemento no **topo** da pilha –  **$O(1)$**
- **stack.pop()** – **Remove** o elemento do **topo** da pilha –  **$O(1)$**
- **s.top()** – **Acessa** o elemento do **topo** da pilha (**sem remover**) –  **$O(1)$**
- **s.empty()** – **Verifica** se a **pilha** está **vazia** (retorna **true** ou **false**) –  **$O(1)$**
- **s.size()** – **Retorna** o **número** de **elementos** na pilha –  **$O(1)$**
- **s.swap(t)** – **Troca** os **elementos** da pilha atual com os de outra pilha –  **$O(1)$**

# Template <stack>

- push(value)



```
1 stack<int> pilha;  
2  
3 pilha.push(10);  
4 pilha.push(20);
```



```
1 stack<int> pilha;  
2  
3 for (int i = 10; i <= 50; i+=10) {  
4     pilha.push(i);  
5 }
```

# Template <stack>

- pop()



```
1  stack<int> pilha;  
2  
3  for (int i = 10; i <= 50; i+=10) pilha.push(i);  
4  // pilha: [50, 40, 30, 20, 10]  
5  
6  pilha.pop(); // pilha: [40, 30, 20, 10]  
7  pilha.pop(); // pilha: [30, 20, 10]
```



# Template <stack>

- `top()`



```
1  for (int i = 10; i <= 50; i+=10) pilha.push(i);
2  // pilha: [50, 40, 30, 20, 10]
3
4  pilha.top(); // 50
5
6  pilha.pop(); // pilha: [40, 30, 20, 10]
7  pilha.top(); // 40
```

# Template <stack>

- `empty()`



```
1  stack<int> pilha;  
2  
3  cout << ((pilha.empty()) ? "pilha vazia" : "pilha não vazia") << "\n";  
4  
5  for (int i = 10; i <= 50; i+=10) pilha.push(i);  
6  
7  cout << ((pilha.empty()) ? "pilha vazia" : "pilha não vazia") << "\n";
```

# Template <stack>

- size()



```
1  stack<int> pilha;  
2  
3  cout << "Quantidade de elementos: " << pilha.size() << "\n";  
4  
5  for (int i = 10; i <= 50; i+=10) pilha.push(i);  
6  
7  cout << "Quantidade de elementos: " << pilha.size() << "\n";
```

# Template <stack>

- **swap(t)** – t é outra pilha

```
1  stack<int> pilhaA, pilhaB;
2
3  for (int i = 10; i <= 50; i+=10) pilhaA.push(i);
4  for (int i = -10; i >= -50; i-=10) pilhaB.push(i);
5
6  pilhaA.swap(pilhaB); // Troca o conteúdo das pilhas
7
8  while (!pilhaA.empty()) { // Percorre a pilha A imprimindo os elementos
9      cout << pilhaA.top() << " ";
10     pilhaA.pop();
11 }
12 cout << "\n"; // -50 -40 -30 -20 -10
```

# Template stack

## Vantagens

- Crescimento sem realocação
- Inserções/remoções eficientes
  - **$O(1)$**
- Menor risco de cópia em crescimento
- Menos fragmentação de memória
- Flexibilidade nas extremidades

## Desvantagens

- Menor desempenho devido ao overhead de memória
- Mais operações de alocação e desalocação
- Complexidade de implementação e manutenção

# Template <stack>

## Alternativas

- **stack** com implementação base: **vector**



```
1 stack<int> pilha; // Por default é implementado com <deque>
2 stack<int, vector<int>> pilha; // Implementação com base no <vector>
```

# Resolução do Problema Motivador

## 1068 – Balanço de Parênteses I

### Dicas:

- Usar uma pilha de char
- Guardar na pilha apenas o '('
- Remover elemento da pilha apenas quando encontrar ')'
- Cuidado, verifique antes de remover da pilha se está vazia
- Usar uma variável Booleana para saber quando a expressão ficou incorreta

A resolução estará disponível no Drive. Tente resolver por conta própria e, se precisar, compare com a solução! 😊

**Template** ◀ **QUEUE** ▶



# Apresentação Problema Motivador

beecrowd | 1110

## Jogando Cartas Fora

Folclore, adaptado por Piotr Rudnicki 🇨🇦 Canada

Timelimit: 1

Dada uma pilha de  $n$  cartas enumeradas de 1 até  $n$  com a carta 1 no topo e a carta  $n$  na base. A seguinte operação é realizada enquanto tiver 2 ou mais cartas na pilha.

Jogue fora a carta do topo e mova a próxima carta (a que ficou no topo) para a base da pilha.

Sua tarefa é encontrar a sequência de cartas descartadas e a última carta remanescente.

Cada linha de entrada (com exceção da última) contém um número  $n \leq 50$ . A última linha contém 0 e não deve ser processada. Cada número de entrada produz duas linhas de saída. A primeira linha apresenta a sequência de cartas descartadas e a segunda linha apresenta a carta remanescente.

### Entrada

A entrada consiste em um número indeterminado de linhas contendo cada uma um valor de 1 até 50. A última linha contém o valor 0.

### Saída

Para cada caso de teste, imprima duas linhas. A primeira linha apresenta a sequência de cartas descartadas, cada uma delas separadas por uma vírgula e um espaço. A segunda linha apresenta o número da carta que restou. Nenhuma linha tem espaços extras no início ou no final. Veja exemplo para conferir o formato esperado.



7  
19  
10  
6  
0

```
Discarded cards: 1, 3, 5, 7, 4, 2  
Remaining card: 6  
Discarded cards: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 4, 8,  
12, 16, 2, 10, 18, 14  
Remaining card: 6  
Discarded cards: 1, 3, 5, 7, 9, 2, 6, 10, 8  
Remaining card: 4  
Discarded cards: 1, 3, 5, 2, 6  
Remaining card: 4
```

**1110 – Jogando**  
**Cartas Fora**

## **Template** < QUEUE > Ou Filas

**As filas são estruturas de dados que seguem a ordem de inserção e remoção FIFO (First In First Out), isso significa que elementos que são inseridos primeiro devem ser os primeiros a serem removidos.**

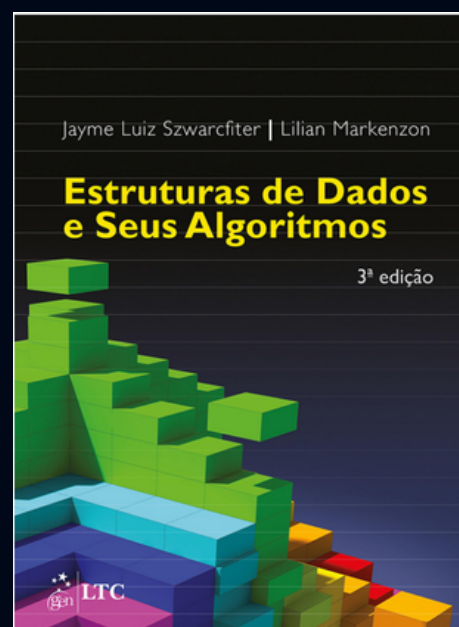
# Template < QUEUE > Ou Filas

## ■ Algoritmo 2.9 Inserção na fila $F$

```
prov := r mod M + 1
se prov ≠ f então
    r := prov
    F[r] := novo-valor
se f = 0 então
    f := 1
senão overflow
```

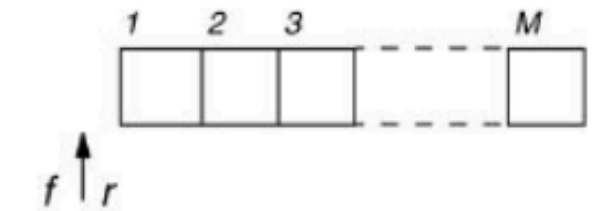
## ■ Algoritmo 2.10 Remoção da fila $F$

```
se f ≠ 0 então
    valor-recuperado := F[f]
se f = r então
    f := r := 0
senão f := f mod M + 1
senão underflow
```



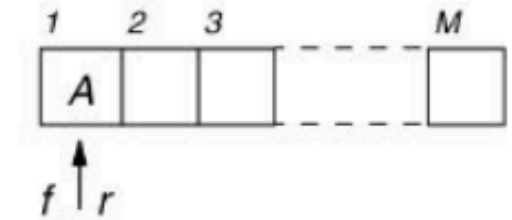
situação 1:

inicial : fila vazia



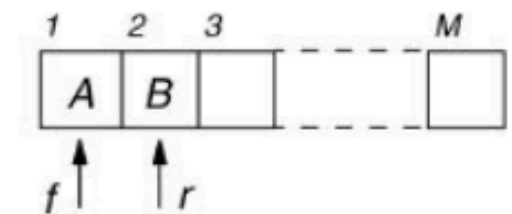
situação 2:

inserir informação A



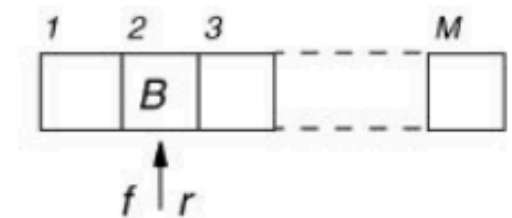
situação 3:

inserir informação B



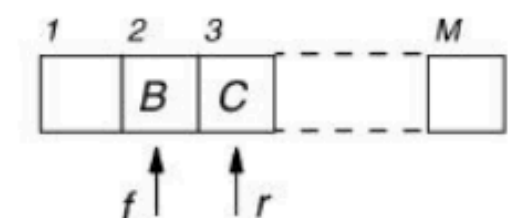
situação 4:

retirar informação (A)



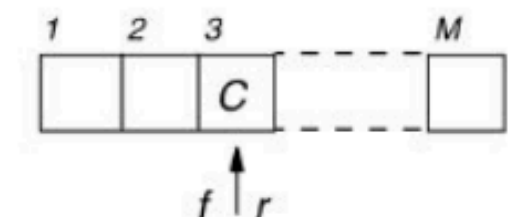
situação 5:

inserir informação C



situação 6:

retirar informação (B)



situação 7:

retirar informação (C)

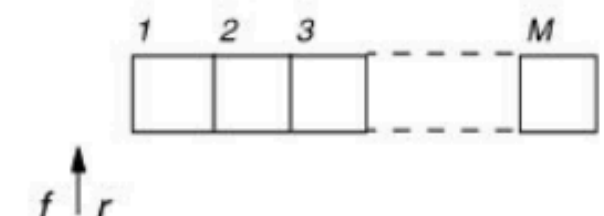


FIGURA 2.3 Operações em uma fila.

## **Template** < QUEUE > Ou Filas

- **CPU scheduling, Disk Scheduling (escalonamento de CPU e disco)**
- **Quando dados são transferidos de forma assíncrona entre dois processos. As filas são usadas para sincronização. Por exemplo: IO Buffers, pipes, file IO, etc**
- **tratamento de interrupções em sistemas de tempo real.**
- **Sistemas de atendimento ao cliente usam filas para organizar as pessoas que ligam para eles.**

# Template <QUEUE> Declaração

```
#include <queue>
```

```
queue<type> q;
```

```
// create a queue of integer data type  
queue<int> integer_queue;
```

```
// create a queue of string data type  
queue<string> string_queue;
```

## Template <QUEUE> Metodos

**push()** – Insere um elemento no fim da fila.

**pop()** – Remove um elemento do começo da fila.

**front()** – Retorna o primeiro elemento da fila.

**back()** – Retorna o ultimo elemento da fila.

**size()** – retorna o numero de elementos na fila.

**empty()** – Retorna True se a fila esta vazia.

# Template <QUEUE> Inserção de Elementos

```
// create a queue of string
queue<string> animals;

// push elements into the queue
animals.push("Cat");
animals.push("Dog");

cout << "Queue: ";

// print elements of queue
// loop until queue is empty
while(!animals.empty()) {

    // print the element
    cout << animals.front() << ", ";

    // pop element from the queue
    animals.pop();
}

cout << endl;
```

# Template <QUEUE> Removendo Valores

```
// create a queue of string
queue<string> animals;

// push element into the queue
animals.push("Cat");
animals.push("Dog");
animals.push("Fox");

cout << "Initial Queue: ";
display_queue(animals);

// remove element from queue
animals.pop();

cout << "Final Queue: ";
display_queue(animals);
```

```
// utility function to display queue
void display_queue(queue<string> q) {
    while(!q.empty()) {
        cout << q.front() << ", ";
        q.pop();
    }

    cout << endl;
}
```



# Template <QUEUE> Acessando Valores

```
// create a queue of int
queue<int> nums;

// push element into the queue
nums.push(1);
nums.push(2);
nums.push(3);

// get the element at the front
int front = nums.front();
cout << "First element: " << front << endl;

// get the element at the back
int back = nums.back();
cout << "Last element: " << back << endl;

return 0;
```

# Template <QUEUE> Verificando Tamanho e Se está vazio

```
// create a queue of string
queue<string> languages;

// push element into the queue
languages.push("Python");
languages.push("C++");
languages.push("Java");

// get the size of the queue
int size = languages.size();

cout << "Size of the queue: " << size;
```

```
// create a queue of string
queue<string> languages;

cout << "Is the queue empty? ";

// check if the queue is empty
if (languages.empty()) {
    cout << "Yes" << endl;
}
else {
    cout << "No" << endl;
}

cout << "Pushing elements..." << endl;

// push element into the queue
languages.push("Python");
languages.push("C++");

cout << "Is the queue empty? ";

// check if the queue is empty
if (languages.empty()) {
    cout << "Yes";
}
else {
    cout << "No";
}
```

**Template** <DEQUE>

# **Template** <DEQUE> Ou Fila Duplamente Terminada

**O deque é um tipo de fila onde podemos fazer a inserção e remoção de elementos em ambas as pontas**

# Template <DEQUE> Declaração

```
#include <deque>
```

```
deque<type> dq;
```

```
// create a deque of integer data type  
deque<int> dq_integer;
```

```
// create a deque of string data type  
deque<string> dq_string;
```

```
// method 1: initializer list  
deque<int> deque1 = {1, 2, 3, 4, 5};
```

```
// method 2: uniform initialization  
deque<int> deque2 {1, 2, 3, 4, 5};
```

## Template <DEQUE> Metodos

**push\_back()** Insere elemento no final  
**push\_front()** Insere elemento no começo  
**pop\_back()** Remove elemento do final  
**pop\_front()** Remove elemento do começo  
**front()** Retorna o elemento que está no começo  
**back()** Retorna o elemento que está no fim  
**at()** Retorna um elemento em um index específico  
**size()** Retorna o numero de elementos  
**empty()** Retorna True se o DEQUE está vazio

# Template <DEQUE> Insertar elementos

```
deque<int> nums {2, 3};

cout << "Initial Deque: ";
for (const int& num : nums) {
    cout << num << ", ";
}

// add integer 4 at the back
nums.push_back(4);

// add integer 1 at the front
nums.push_front(1);

cout << "\nFinal Deque: ";
for (const int& num : nums) {
    cout << num << ", ";
}
```

# Template <DEQUE> Acessando elementos

```
deque<int> nums {1, 2, 3};

cout << "Front element: " << nums.front() << endl;
cout << "Back element: " << nums.back() << endl;
cout << "Element at index 1: " << nums.at(1) << endl;
cout << "Element at index 0: " << nums[0];
```

```
deque<int> nums = {1, 2};

cout << "Initial Deque: ";

for (const int& num : nums) {
    cout << num << ", ";
}

// change elements at the index
nums.at(0) = 3;
nums.at(1) = 4;

cout << "\nUpdated Deque: ";

for (const int& num : nums) {
    cout << num << ", ";
}
```



# Template <DEQUE> Removendo elementos

```
deque<int> nums {1, 2, 3};

cout << "Initial Deque: ";
display_deque(nums);

// remove element from the back
nums.pop_back();

cout << "\nDeque after pop_back(): ";
display_deque(nums);

// remove element from the front
nums.pop_front();

cout << "\nDeque after pop_front(): ";
display_deque(nums);
```

```
// utility function to print deque elements
void display_deque(deque<int> dq){
    for (const int& num : dq) {
        cout << num << ", ";
    }
}
```

# Template <DEQUE> Usando Iterators

```
deque<int> nums {1, 2, 3};

// declare an iterator to deque
deque<int>::iterator dq_iter;

// make iterator point to first element
dq_iter = nums.begin();

// print value pointed by iterator using *
int first_element = *dq_iter;

cout << "nums[0] = " << first_element << endl;

// make iterator point to element at index 1
dq_iter = nums.begin() + 1;
int element_index1 = *dq_iter;

cout << "nums[1] = " << element_index1 << endl;

// make iterator point to last element
dq_iter = nums.end() - 1;
int last_element = *dq_iter;

cout << "nums[2] = " << last_element;
```

# Resolução do Problema Motivador

A resolução estará disponível no Drive. Tente resolver por conta própria e, se precisar, compare com a solução! 😊

# Lista de Exercícios

## 1068 – Balanço de Parênteses I



Se tiver alguma dúvida ou dificuldade na resolução de algum exercício, sinta-se à vontade para perguntar! 😊

# Referências

**[1] JAVA RUSH. Diagrama de Red-Black Tree** [imagem]. Disponível em: <https://cdn.javarush.com/images/article/9a5b5d15-c32b-4b6f-9f8e-b1d12908379c/1080.jpeg>. Acesso em: 03 abr. 2025.

**[2] PROGRAMIZ. C++ Unordered Set.** Disponível em: <https://www.programiz.com/cpp-programming/unordered-set>. Acesso em: 03 abr. 2025.

**[3] PROGRAMIZ. Hash Table.** Disponível em: <https://www.programiz.com/dsa/hash-table>. Acesso em: 03 abr. 2025.

**[4] GLASIELLY DEMORI. Árvore Rubro-Negra (Aula 11) – Inserção.** YouTube, 10 out. 2023. Disponível em: <https://www.youtube.com/watch?v=vSAE4O2zpkY>. Acesso em: 4 abr. 2025.