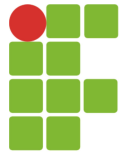
Filas

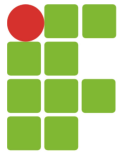
Introdução à Programação Competitiva

Prof. Daniel Saad Nogueira Nunes

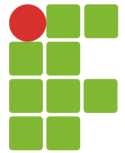
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário



Sumário



Introdução

- A Fila é um **tipo abstrato de dados** que possui a propriedade FIFO (first-in-first-out).
- Isto é, os primeiros elementos que são inseridos devem ser os primeiros a serem retirados.
- **Idealmente** as operações de inserção na cauda da fila, remoção da frente da fila e acesso à frente da fila, devem ser realizadas em tempo $\Theta(1)$.

Introdução

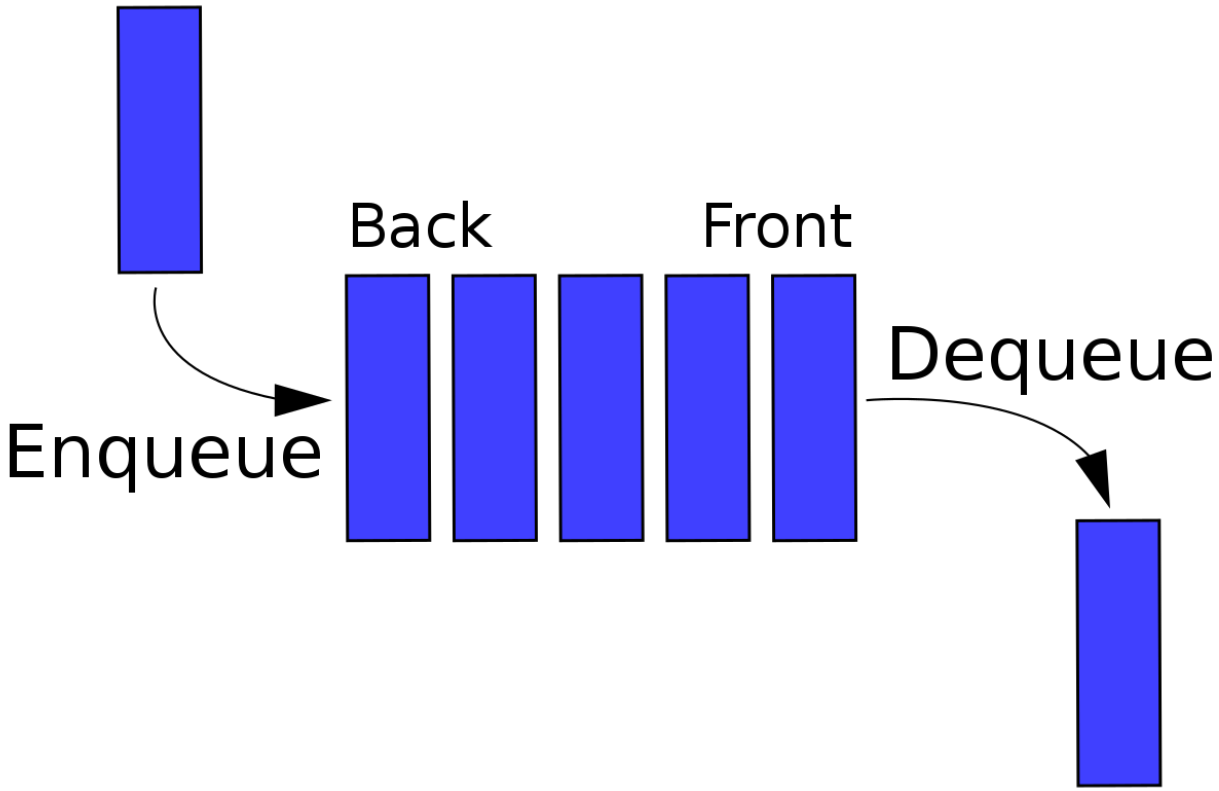
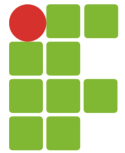
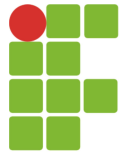


Figura: Operações em filas. Fonte:
`https://upload.wikimedia.org/wikipedia/commons/thumb/5/52/Data_Queue.svg/220px-Data_Queue.svg.png`



Implementação do TAD

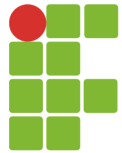
- Filas podem ser implementadas de diversas formas.
- Duas das formas são através de:
  - Listas autorreferenciadas.
  - Vetores dinâmicos



Implementação do TAD

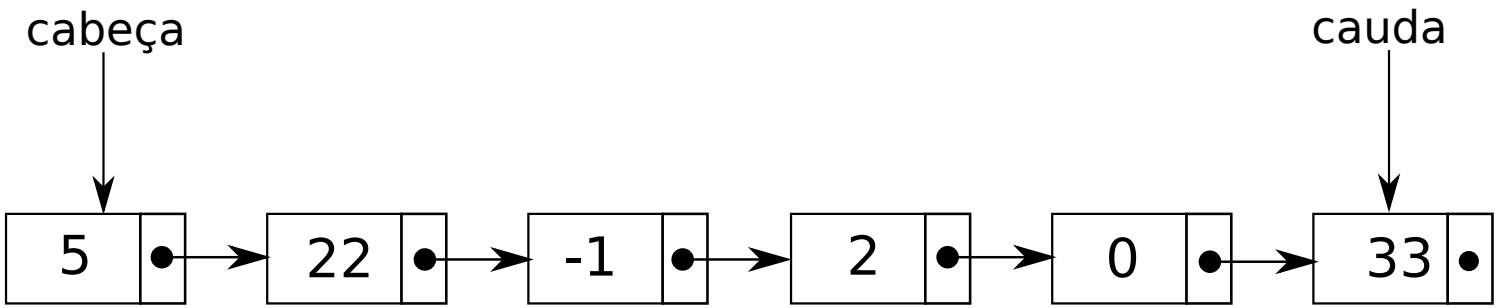
Listas autorreferenciadas

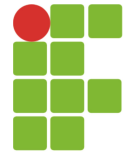
- Para suportar a operação de inserção na cauda da fila, basta inserir na cauda da lista.
- Para suportar a operação de remoção da frente da fila, basta remover na cabeça da lista.
- O acesso à frente equivale ao acesso à cabeça da lista.



Implementação do TAD: listas autorreferenciadas

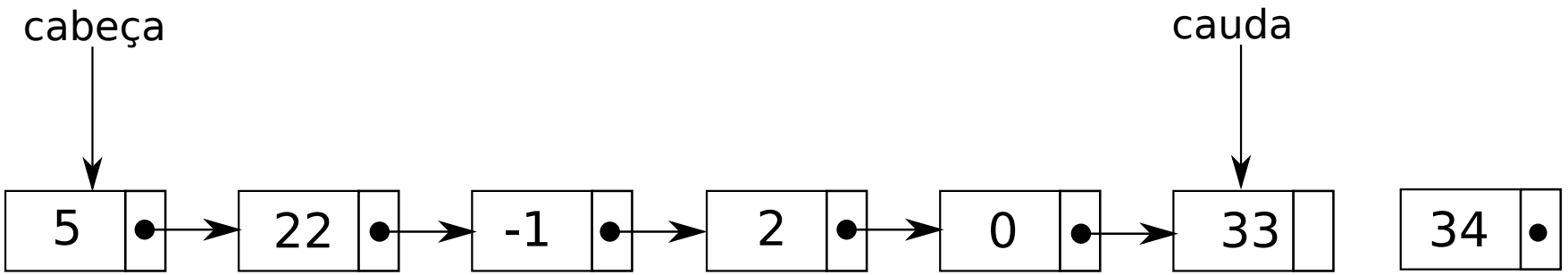
Inserção

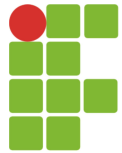




Implementação do TAD: listas autorreferenciadas

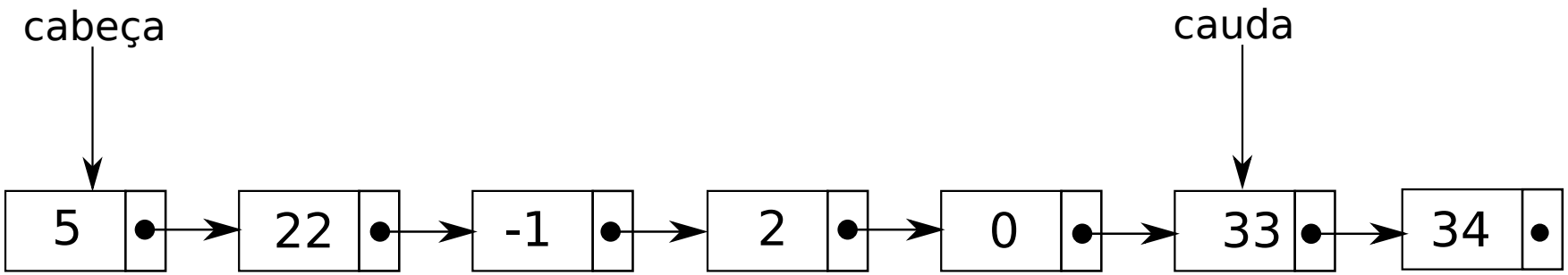
Inserção

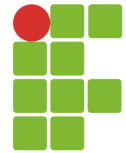




Implementação do TAD: listas autorreferenciadas

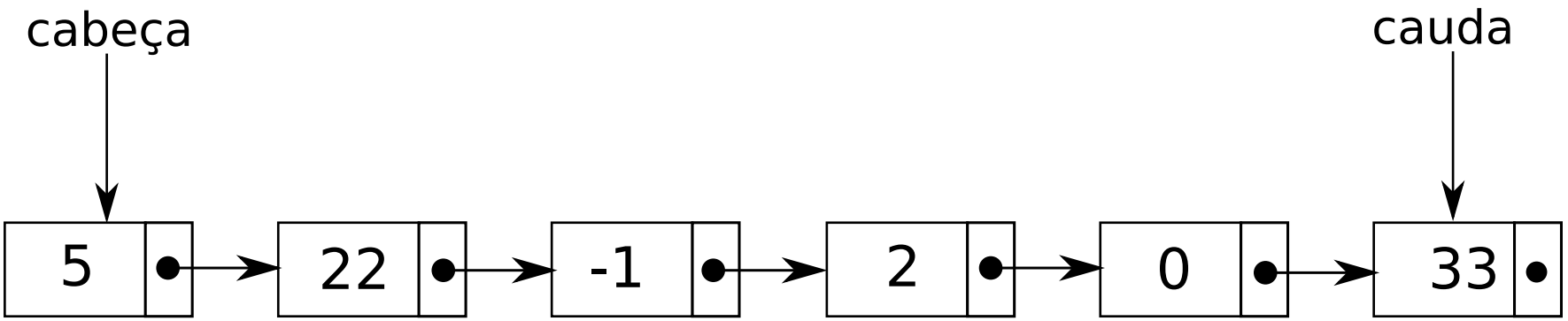
Inserção

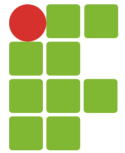




Implementação do TAD: listas autorreferenciadas

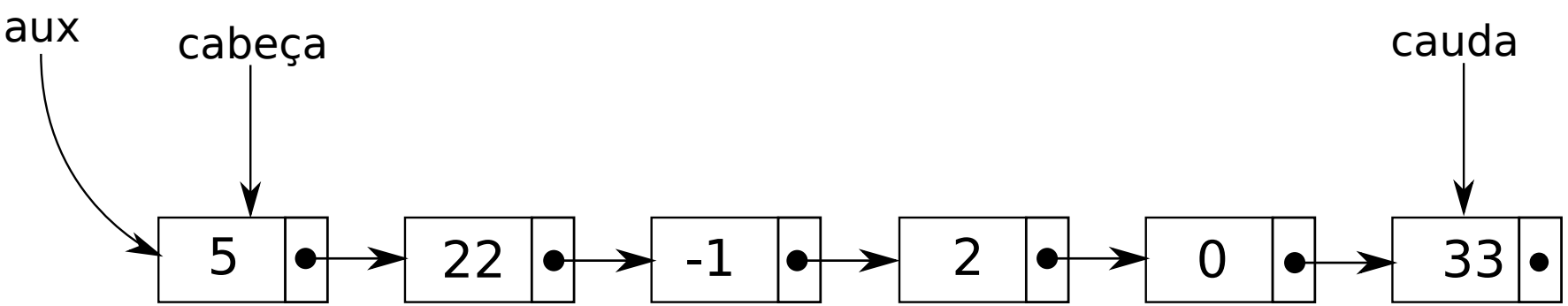
Remoção

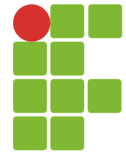




Implementação do TAD: listas autorreferenciadas

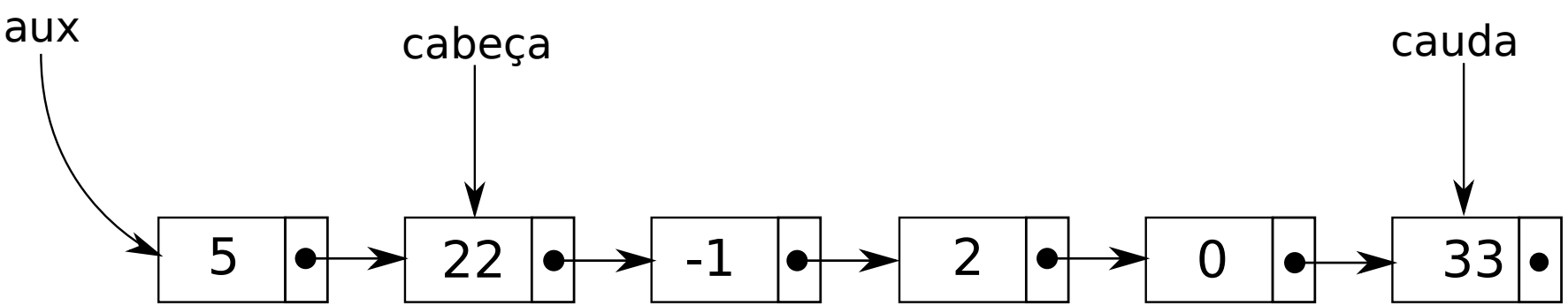
Remoção

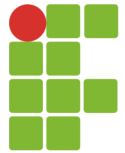




Implementação do TAD: listas autorreferenciadas

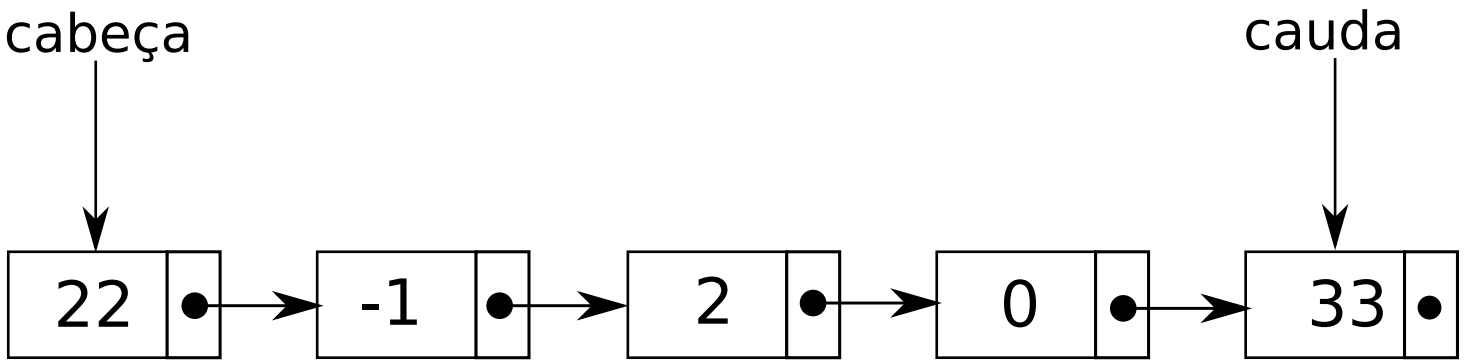
Remoção

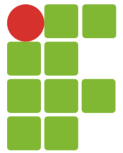




Implementação do TAD: listas autorreferenciadas

Remoção





Implementação do TAD

Vetores dinâmicos

- Na implementação utilizando vetores dinâmicos utilizamos dois índices, `rear` e `front` para armazenar a primeira e última posições da fila.
- Sempre que um elemento precisa ser inserido, incrementa-se a variável `rear` e o elemento é inserido nesta posição.
- Sempre que um elemento precisa ser removido, incrementa-se a variável `front`
- Para reutilizar o espaço do vetor, trabalhamos com ele de maneira circular, o que pode ser feito com o operador de resto.
- Caso a fila fiquei cheia, redimensionamos o vetor.

Implementação do TAD: vetores dinâmicos

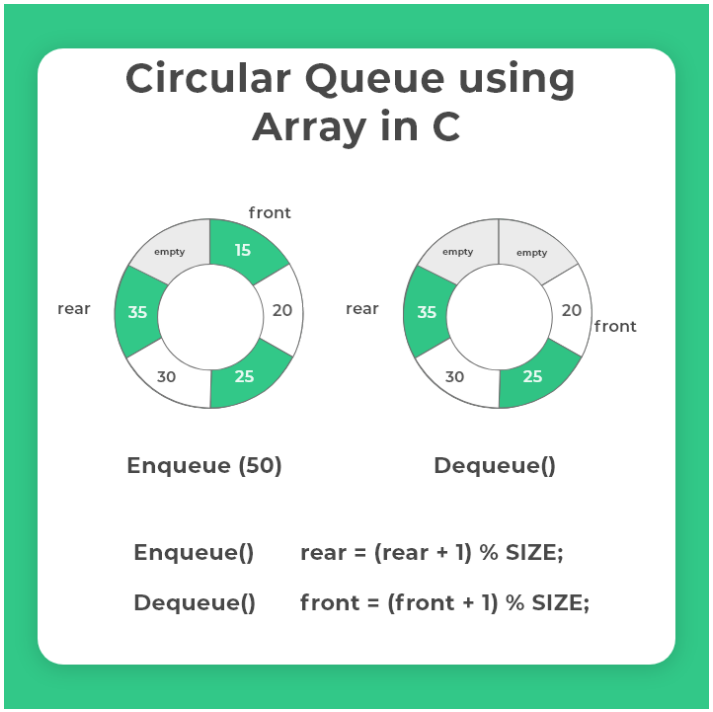
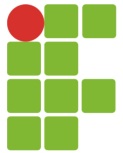
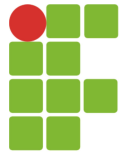


Figura: Filas em vetores circulares. Fonte: `https://prepinsta.com/data-structures-algorithms/circular-queue-using-array-in-c/`

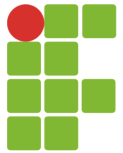


Sumário

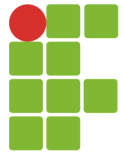


Filas: C++

- A STL do C++ fornece o tipo fila totalmente parametrizável.
- Por padrão é implementada através de um `std::deque`, mas o *container* `std::list` pode ser utilizado.
- Inserção, remoção e acesso à frente em tempo $\Theta(1)$!
- Cabeçalho: `<queue>`.

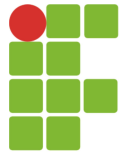


Sumário

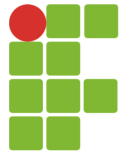


Declaração

- Para declarar uma pilha, utilizamos: `std::queue<T> nome_variavel;`, em que `<T>` corresponde ao tipo desejado.
- Exemplos:
  - `std::queue<int> fila_int;`
  - `std::queue<pessoa> fila_pessoa;`
  - `std::queue<std::pair<int,string>> fila_par;`
  - `std::queue<vector<int>> fila_vetor;`

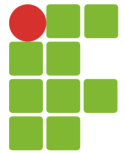


Sumário

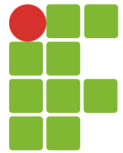


Inserção

- Para inserir na cauda da fila, utilizamos o método `push`.
- O `emplace` também pode ser utilizado no caso de uma inserção *in-place* (`C++11`).
- Tempo $\Theta(1)$.

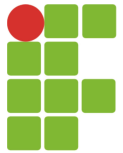


Sumário

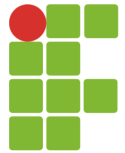


Remoção

- Para remover o item da frente, utilizamos o `pop`.
- Tempo $\Theta(1)$.

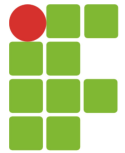


Sumário

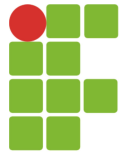


Acesso

- O acesso ao elemento da frente da fila é realizado através do método `front`.
- Também é possível acessar o último elemento da fila através do método `back`
- Tempo $\Theta(1)$.

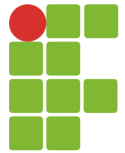


Sumário

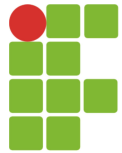


Limpeza

- O método `clear` pode ser utilizado para deletar todos os elementos de uma pilha.

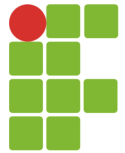


Sumário



Métodos auxiliares

- `bool empty() const;` : retorna verdadeiro se a fila está vazia.
- `size_t size() const;` : retorna o tamanho da fila.



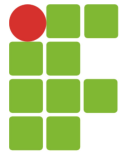
Exemplo

Jogando Cartas Foras (URI 1110)

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.

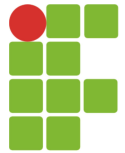
Sua tarefa é encontrar a sequência de cartas descartadas e a última carta remanescente.



Exemplo

Entrada

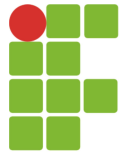
Cada linha de entrada (com exceção da última) contém um número n. A última linha contém 0 e não deve ser processada.



Exemplo

Saída

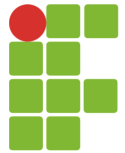
Cada número de entrada produz duas linhas de saída. A primeira linha apresenta a sequência de cartas descartadas e a segunda linha apresenta a carta remanescente.



Exemplo

Restrições

- $1 \leq n \leq 50$



Exemplo

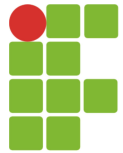
Exemplo de entrada/saída

- Entrada:

```
7
19
10
6
0
```

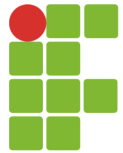
- Saída:

```
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
```



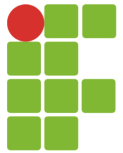
Exemplo

- Para resolver este problema, podemos utilizar uma fila que guarda todos os números de 1 a n nesta ordem.
- Basta então simular o processo descrito: descartamos o primeiro elemento da fila e depois movemos o próximo primeiro elemento no final da fila.
- Repetimos até que a fila fique com tamanho 1.

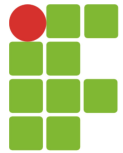


Solução

```
1  #include <iostream>
2  #include <queue>
3  #include <vector>
4  using namespace std;
5
6  int main() {
7      std::ios::sync_with_stdio(false);
8      int n;
9      while (cin >> n && n) {
10         queue<int> q;
11         for (int i = 1; i <= n; i++) {
12             q.push(i);
13         }
14         cout << "Discarded cards: ";
15         while (q.size() > 1) {
16             cout << q.front();
17             q.pop();
18             q.push(q.front());
19             q.pop();
20             cout << (q.size() > 1 ? ", " : "\n");
21         }
22         cout << "Remaining card: " << q.front() << endl;
23     }
24     return 0;
25 }
```



Sumário



Referências

cppreference, *cppreference.com*, `https://en.cppreference.com/`, Acessado em 12/2022.