Pilha e Fila

Conceito e implementação em vetor

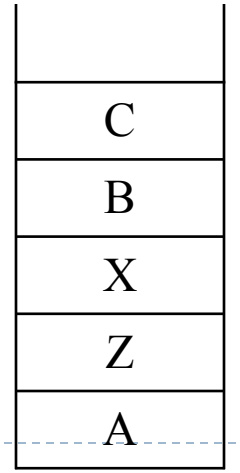
Jacqueline Midlej

Pilha

- Conceito simples
- Porém muito utilizado nas áreas de computação como ferramenta para solução de problemas
  - Lembra da pilha de execução de um programa??
  - O que ocorre quando chamamos funções dentro de outras funções?
  - Ou mesmo, o que ocorre quando chamamos funções recursivamente?

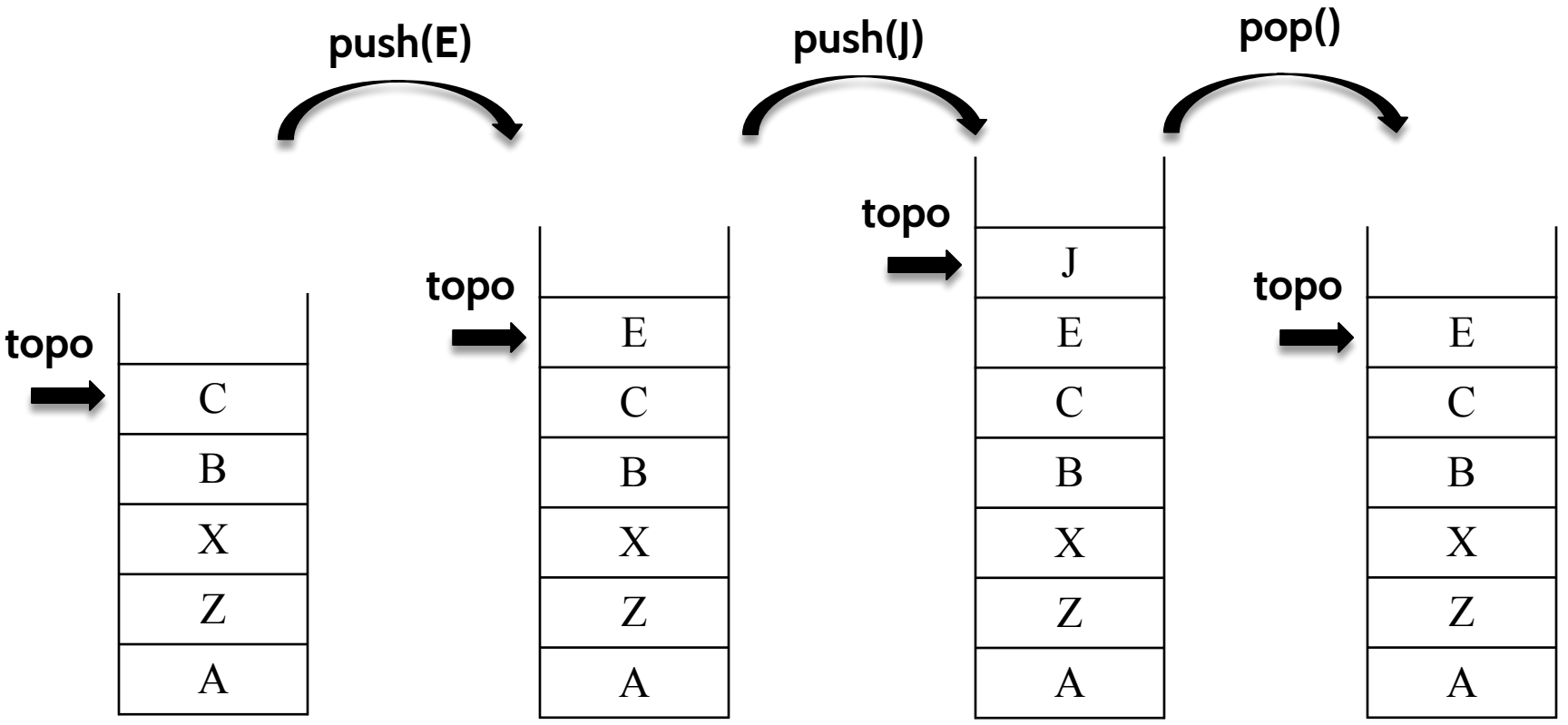
Definição

- Conjunto ordenado de itens
- Elementos são inseridos e removidos de uma única extremidade da pilha: o **topo**
- Compreende operações de inserção e eliminação de itens de forma dinâmica, constantemente mutável



Definição

- Regra **LIFO – Last-in, First-out**
  - Novos itens são colocados no topo da pilha
    - Operação chamada de empilhar: **push(s,i)**
    - O topo deve ser descolado
  - Ou os itens do topo são removidos
    - Operação chamada de desempilhar: **pop(s)**
    - A operação deve retornar o elemento desempilhado
    - O topo deve ser deslocado para baixo
- Desta forma, a pilha expande e reduz com o passar do tempo


push(E)
push(J)
pop()
topo
C
B
X
Z
A
topo
E
C
B
X
Z
A
topo
J
E
C
B
X
Z
A
topo
E
C
B
X
Z
A

Operações

- Não existe limite máximo para o número de itens que podem ser mantidos em uma pilha
- Se a pilha contém um único elemento e ele for desempilhado, resultará em uma **pilha vazia**
  - **push** pode ser aplicado a qualquer pilha
  - **pop** não pode ser aplicado a uma pilha vazia
  - **empty** verificará se a pilha está vazia ou não, devolvendo TRUE ou FALSE (ou 0 e 1)
  - **stacktop** determina/retorna o item superior da pilha, sem removê-lo
    - Não pode ser aplicado para pilha vazia
    - Pode ser composta de duas operações básicas, quais são?

Exemplo de uso de uma pilha

7- ((X * ((X+Y) / (J-3)) +Y) / (4-2.5))

- Verificação de parênteses:
  - **Regra 1:** O número de parênteses esquerdo [(] e direito [)] são iguais
    - ((A+B) **errado!**
  - **Regra 2:** Todo parâmetro da direita deve ser precedido por um da esquerda
    -)A+B (-C **errado!**
  - Neste problema, cada par de parênteses representa um escopo, onde o parênteses de esquerda abre o escopo e o da direita fecha
  - A profundidade de agrupamento num ponto é a quantidade de parênteses abertos e não fechados nesse ponto

Exemplo de uso de uma pilha

- Solução:
  - Realizar contagem de parênteses (CP) da seguinte forma:
    - CP= paranteses_esq – parenteses_dir
  - Em um dado ponto, CP representa a profundidade de agrupamento naquele ponto
  - CP deve ser igual a 0 no fim da expressão (satisfaz regra 1)
  - CP em cada ponto é não negativa (satisfaz regra 2)

Exemplo de uso de uma pilha

$$7-((X*((X+Y)/(J-3))+Y)/(4-2.5))$$

0 0 1 2 2 2 3 4 4 4 4 3 3 4 4 4 4 3 2 2 2 1 1 2 2 2 2 1 0

- Se alteramos o problema, supondo a existência de outros tipos diferentes de limitadores. Ex. [] e {}
  - O finalizador de escopo (símbolo da direita) deve ser correspondente ao seu iniciador (símbolo da esquerda)
    - (A+B] [(A+B]) **errado!**

- Precisamos diferenciar os tipos de símbolo, além de diferenciar o lado (esquerdo e direito)

Exemplo de uso de uma pilha

- Solução:
  - Inicia uma pilha e percorre a string com a expressão numérica
  - Sempre que um iniciador de escopo for econtrado, ele será empilhado
  - Sempre que um finalizador de escopo for encontrado a pilha deve ser verificada:
    - Se a pilha estiver vazia: **expressão inválida**
    - Se não, se o item do topo é o inicializador (esquerda) correspondente ao símbolo lido, desempilha e continua
    - Se não corresponde: **expressão inválida**
    - Se ao fim, a pilha estiver vazia: **expressão válida**

Implementação em vetor

- A pilha é uma estrutura de dados abstrata, independe da implementação
- Vamos estudar aqui, uma maneira de implementar essa estrutura abstrata: usando vetor
- Pilha tem tamanho dinâmico, vamos simular isso no vetor:
  - Definimos um tamanho máximo do vetor e controlamos a expansão e compressão da pilha com uma variável (topo)
  - Uma extremidade do vetor será fixa (início) e a outra extremidade será variável (topo)

Implementação em vetor

```
#define STACKSIZE 100
struct stack{
  int top;
  int items[STACKSIZE];
};
struct stack s;
```

- Aqui, definimos e instanciamos uma pilha com no máximo 100 elementos inteiros.
- A variável ***top*** guarda a última posição que contém elemento.
  - Pode-se definir outros tipos de dados para os elementos de uma pilha, inclusive um dado complexo representado por outra estrutura. Ex: um aluno que contém nome e RA.

Implementação em vetor

```
#define STACKSIZE 100
#define NAMESIZE 30

struct student {
   char name[NAMESIZE];
   int ra;
};

struct stack {
   int top;
   struct student items[STACKSIZE];
};
```

Implementação em vetor

- A variável top deve ser sempre um inteiro, por quê?
- Como podemos indicar que a pilha está vazia?

Implementação em vetor

- A variável top deve ser sempre um inteiro, por quê?
- Como podemos indicar que a pilha está vazia?

```
int empty (struct stack *ps){
    if (ps->top = -1)
        return 1; //true
    else
        return 0; //false
}
```

Implementação em vetor

- Porque ter uma função empty ao invés de verificar direto no código quando necessário?

Implementação em vetor

- Porque ter uma função empty ao invés de verificar direto no código quando necessário?

- Resposta
  - Programas mais legíveis/compreensíveis
  - Usa pilha independente da implementação (veremos outras formas de implementar a pilha)
  - Empty tem significado, independente do valor de top (nem todas as implementações usam top=-1 para pilha vazia)

Implementação em vetor

- Pop:
  - Se a pilha estiver vazia, reportar erro!
  - Se não, remove o primeiro elemento da pilha (isto é, o elemento do topo)
  - Retorna esse elemento para o programa/função chamador

Implementação em vetor

```
int pop (struct stack *ps){
    if (empty(ps)){
        printf("%s", "stack underflow");
        exit(1);
    }
    return (pos->items[ps->top--]);
}
```

- Nossa pilha contém valores inteiros;
- E se nossa pilha tivesse alunos? retorno aluno!
  - struct student pop(struct stack *ps);

Implementação em vetor

- Porque passagem de parâmetro por referência?

Implementação em vetor

- Porque passagem de parâmetro por referência?
  - Queremos modificar o valor da variável top
  - Se passarmos apenas o valor, a operação top-- será "perdida" ao encerrar a função

Implementação em vetor

```
void push (struct stack *ps, x){
    ps->items[++(pos->top)]=x;
    return;
}
```

- O que acontece se o número exceder o número máximo?

Implementação em vetor

```
void push (struct stack *ps, x){
    ps->items[++(ps->top)]=x; // primeiro incrementa o topo,
                              //depois coloca o elemento x no topo
    return;
}
```

- O que acontece se o número exceder o número máximo?

```
void push (struct stack *ps, x){
    if (ps->top ==STACKSIZE-1){
        printf("%s", "estouro de pilha");
        exit(1);
    }
    ps->items[++(ps->top)]=x;
    return;
}
```

Exercício 1

- Faça a execução de uma pilha para a sequência de empilhamentos e desempilhamentos abaixo. Mostre as chamadas para as funções e o estado da pilha, onde:
- s é a pilha,
- s.top o topo da pilha,
- x variável para armazenar elemento retornado pela pilha

- Sequência:
  - Empilha 1, Desempilha, Desempilha, Empilha 1, Empilha 2, Empilha 3, Desempilha, Empilha 4, Desempilha, Desempilha, Empilha 5.
- Informe também quando ocorre erros!

Vamos codificar e testar essa pilha?

Exercício 2

- Escreva um pseudocódigo usando pilha que, dada uma string no formato xCy, onde x e y são formados por As e Bs, determine se y é a leitura inversa de x (isto é, x de trás pra frente).
- Exemplo:
  - Para ABBBCBBBA => sim
  - Para ABBBCABBB => não

- NÃO escreva as funções da pilha. Considere que **empty**, **push** e **pop** já são implementadas para um vetor de caracteres

FILA

Difinição

- Conjunto ordenado de itens de tamanho variável
- Regra **FIFO: First-in, First-out**
  - Insere-se elementos no fim da fila
  - Remove elementos do início da fila
- Exemplos: Fila de banco, pedágio (...)

- Operações primitivas:
  - insert(q, x): insere o elemento x na fila q
  - x=remove(q): elimina um elemento da fila q
  - empty(q): verifica se a fica está vazia

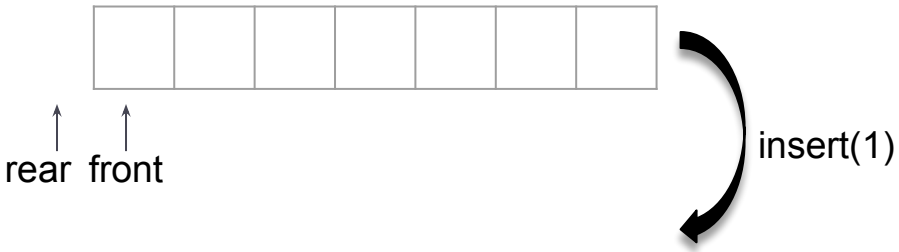
Implementação em vetor

- Usar duas variáveis para controlar o tamanho da fila:
  - **front**, indica o inicio da fila
  - **rear** indica o último da fila

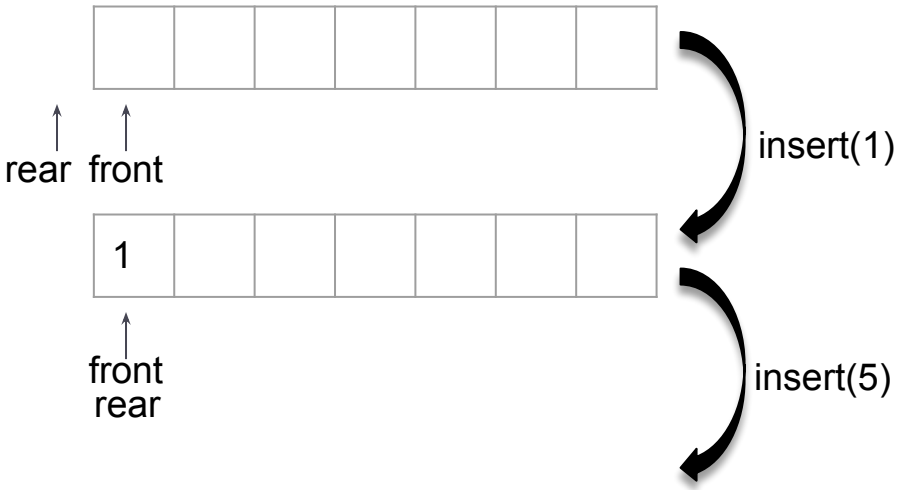
```
#define MAXQUEUE 100
struct queue {
    int items[MAXQUEUE];
    int front, rear;
}
```

  - Ignorando a possibilidade de estouro:
    - insert: q.items[++q.rear]=x;
    - remove: x=q.items[q.front++];
    - Inicialmente q.rear=-1 e q.front=0;
    - empty se q.front > q.rear

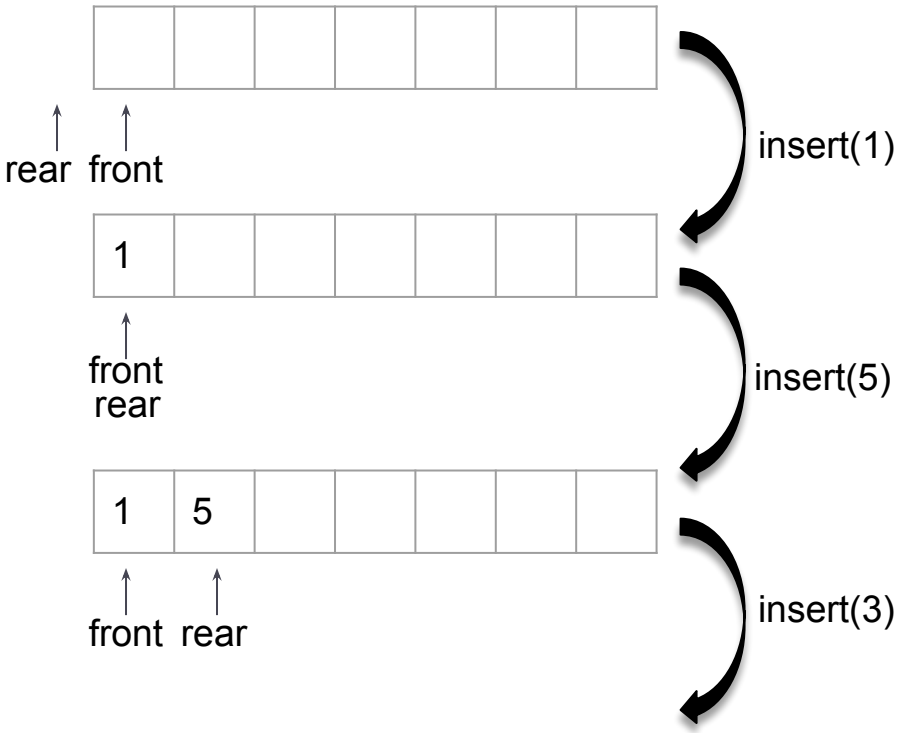
Execução



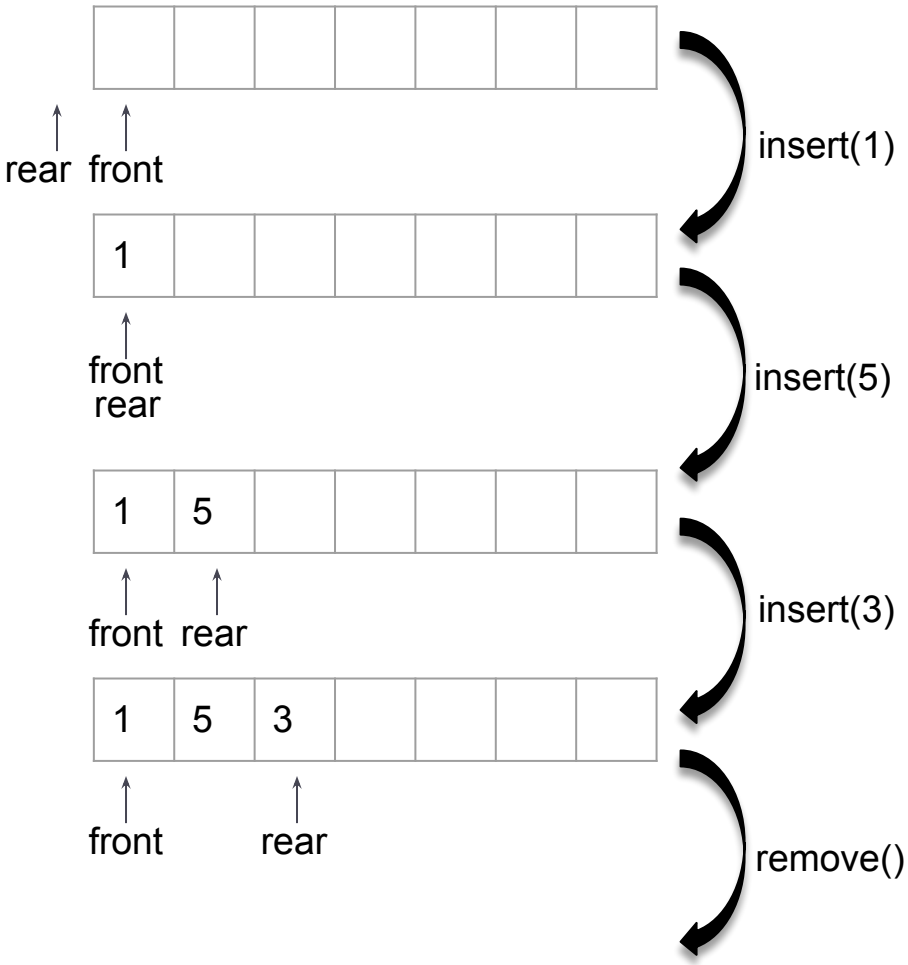
Execução



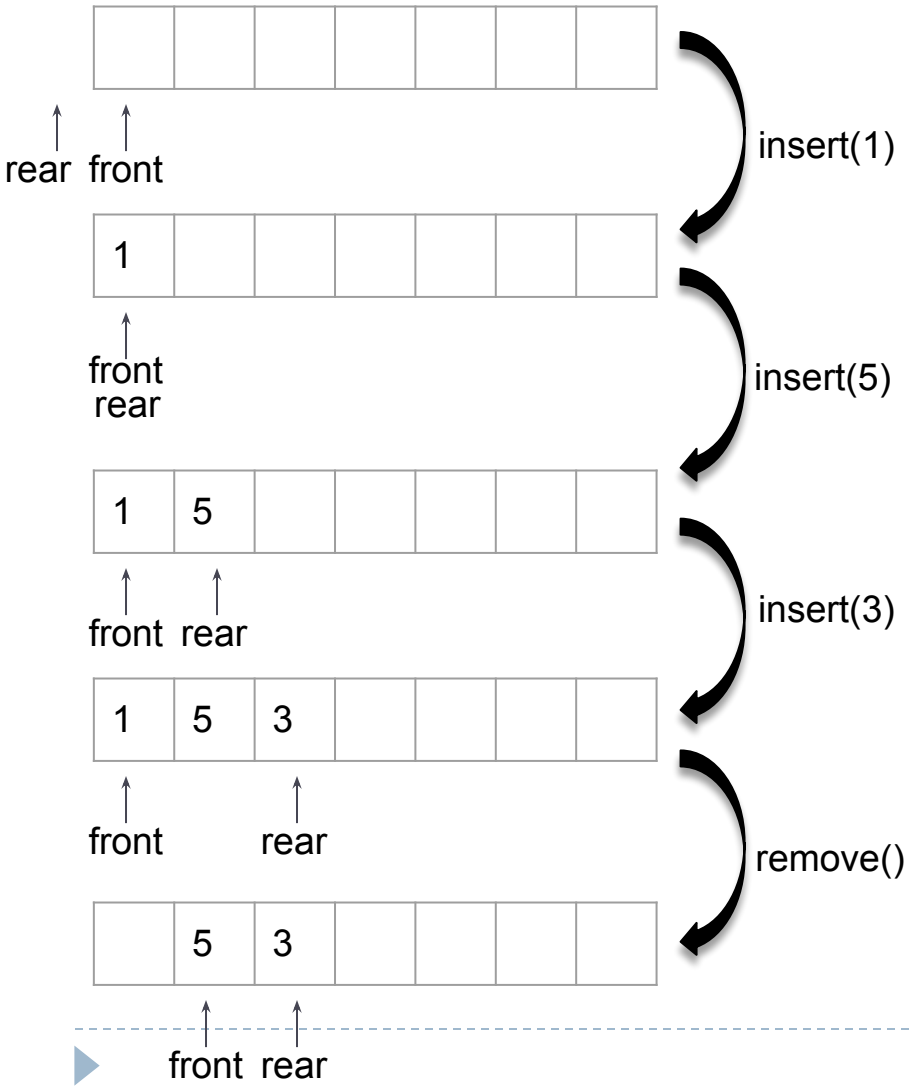
Execução

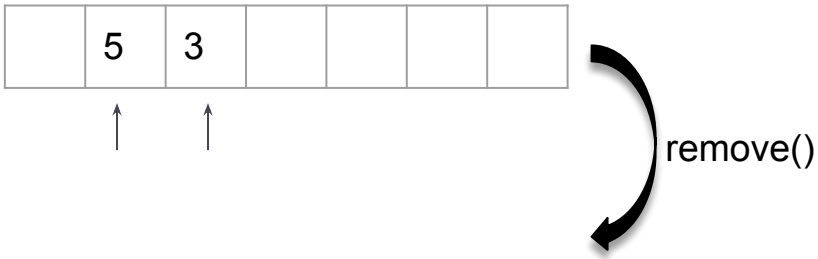

rear
front
insert(1)
1
front
rear
insert(5)
1
5
front
rear
insert(3)

Execução

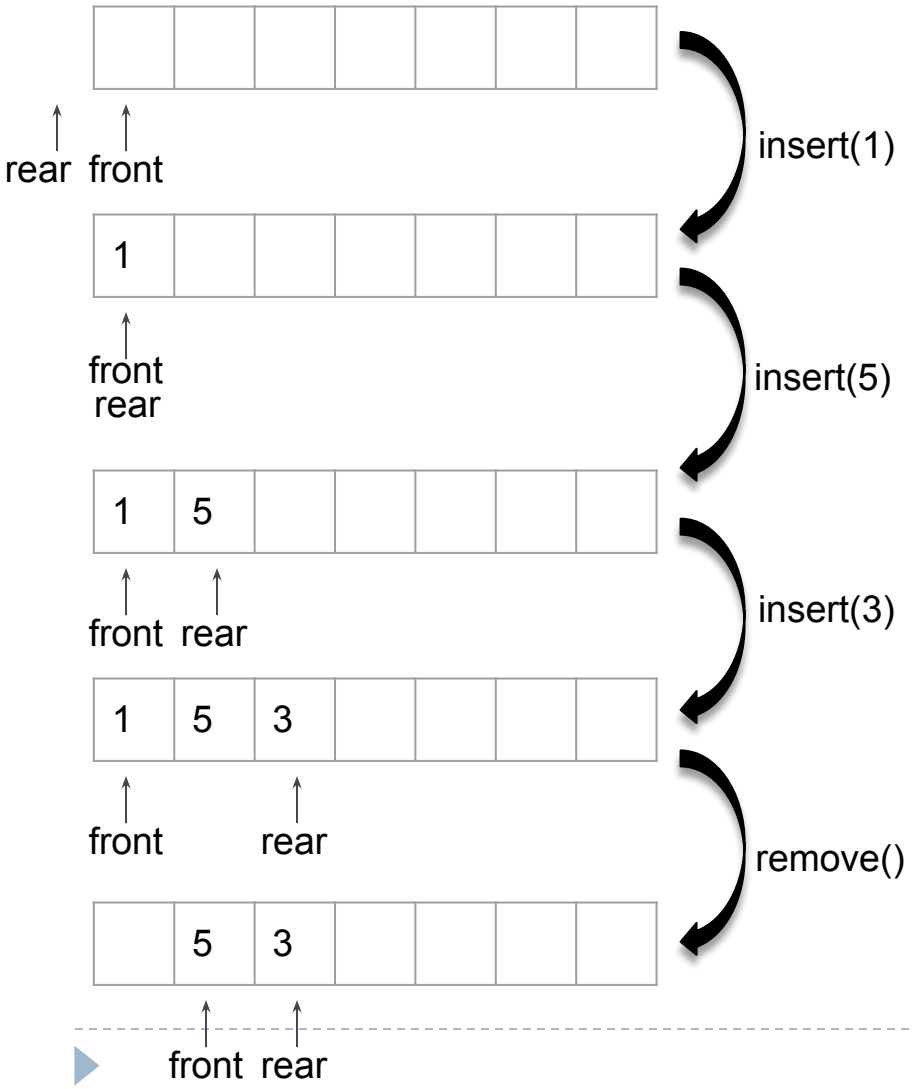

rear
front
insert(1)
1
front
rear
insert(5)
1
5
front
rear
insert(3)
1
5
3
front
rear
remove()

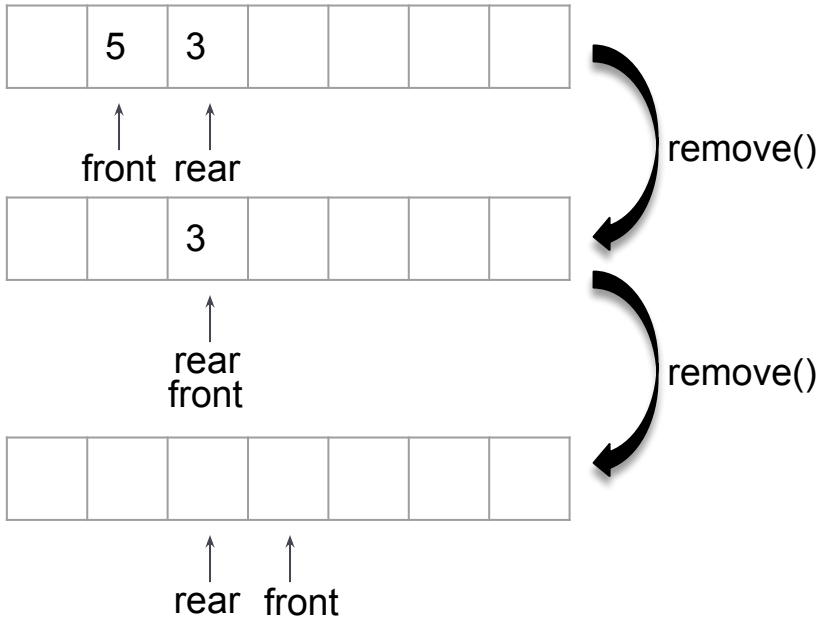
Execução


rear
front
insert(1)
1
front
rear
insert(5)
1
5
front
rear
insert(3)
1
5
3
front
rear
remove()
5
3
front
rear


5
3
remove()

Execução

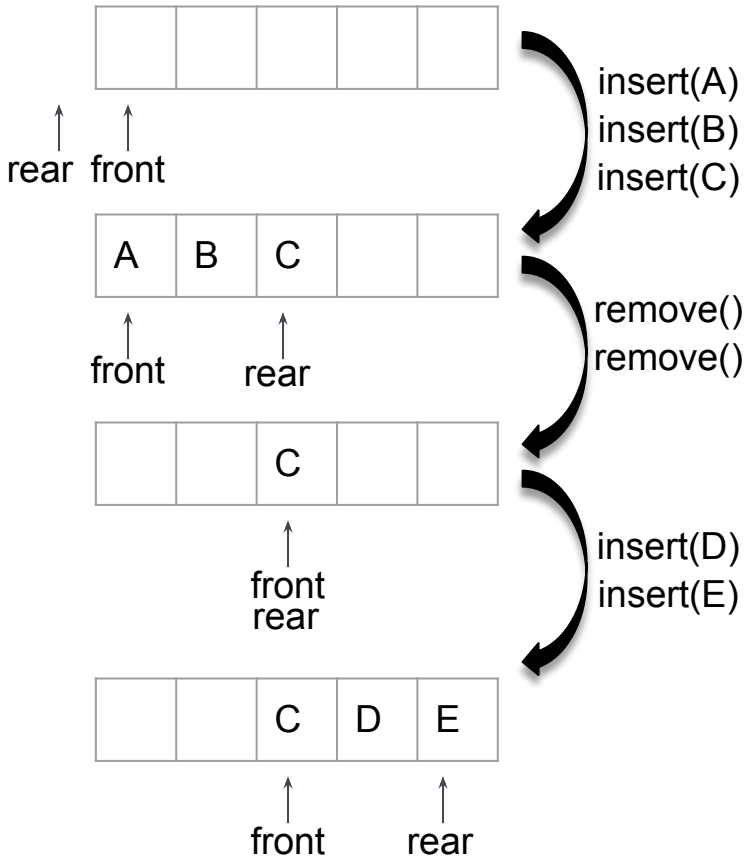

rear
front
insert(1)
1
front
rear
insert(5)
1
5
front
rear
insert(3)
1
5
3
front
rear
remove()
5
3
front
rear


5
3
front
rear
remove()
3
rear
front
remove()
rear
front

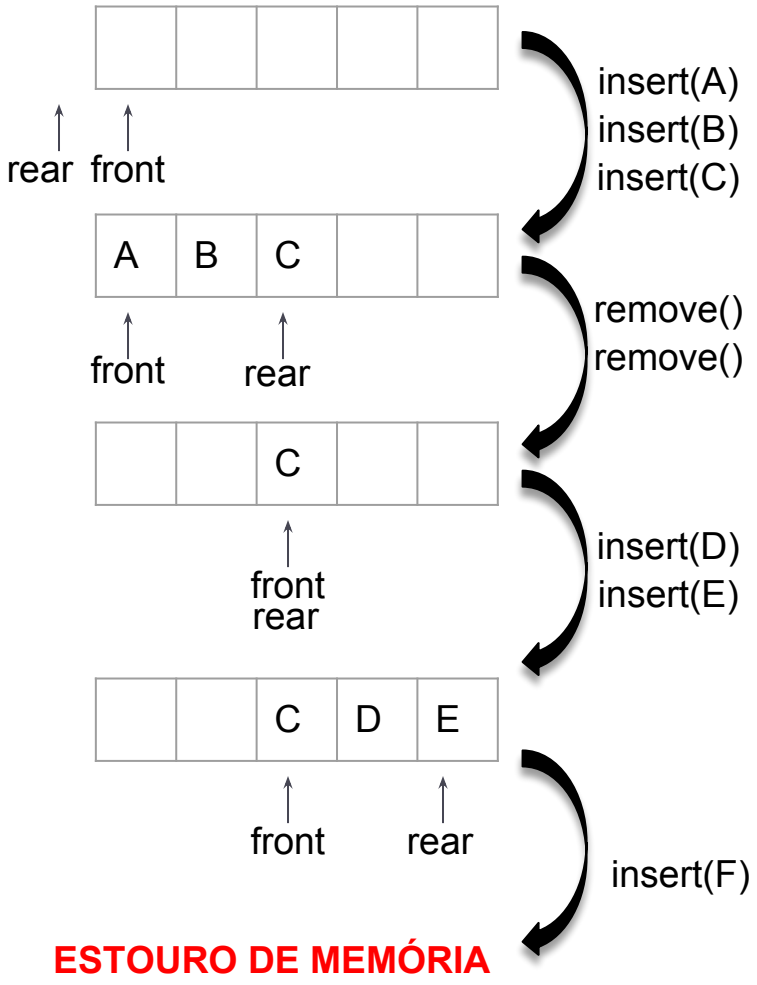
Exercício

- Implemente as operações da fila!!
- Faça um main realizando as seguintes operações em ordem:
  - 1 insert e 1 remove
  - 2 insert e 1 remove
  - 2 insert e 4 de remove

Execução 2 - Problema??



Execução 2 - Problema??



Soluções possíveis

- Deslocar todos os elementos para frente em uma remoção

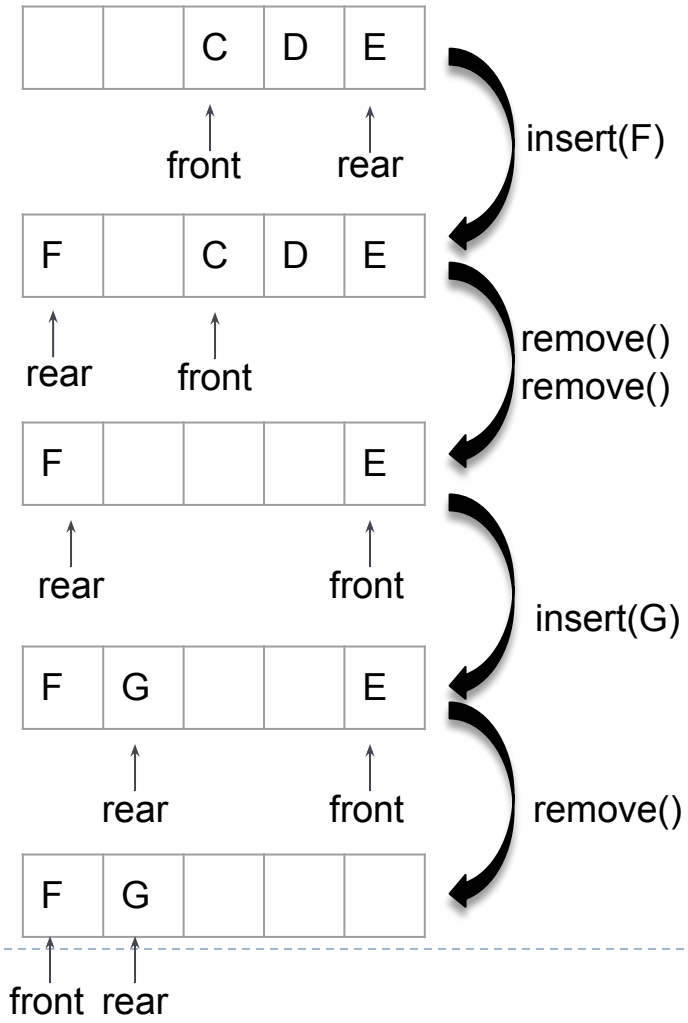
```
x = q.itens[0];
for (i = 0; i< q.rear; i++)
     q.items[i] = q.items [i+1];
q.rear--;
```

  - Manter apenas uma variável para o fim da fila
    - O elemento da posição 0 sempre será o primeiro da fila

- Problema: Ineficiente!

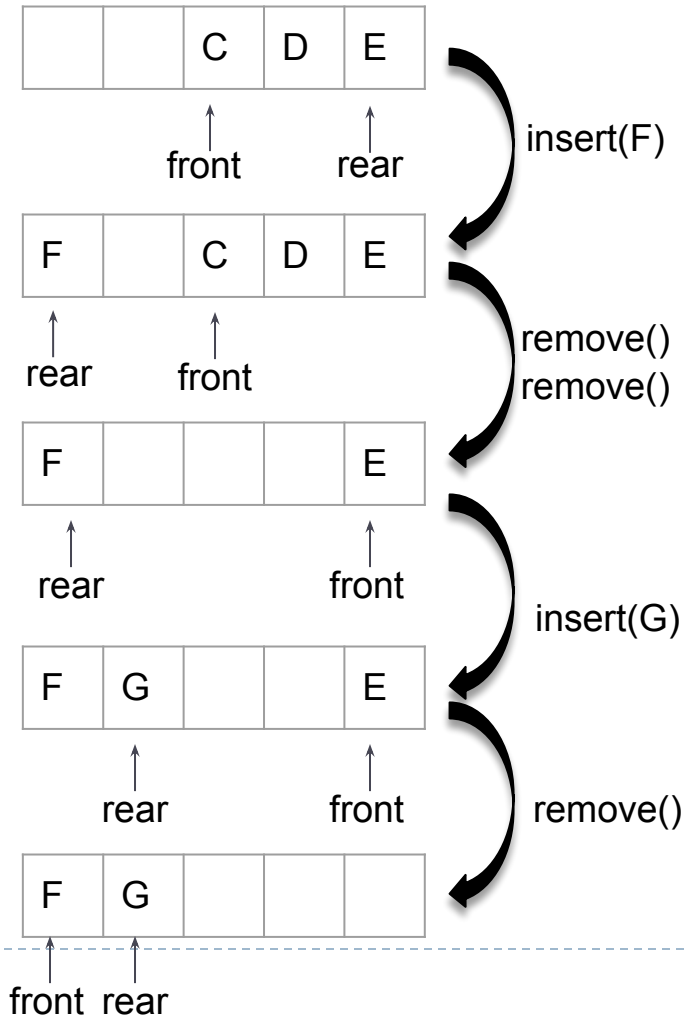
Soluções possíveis

- Visualizar o vetor como uma lista circular!



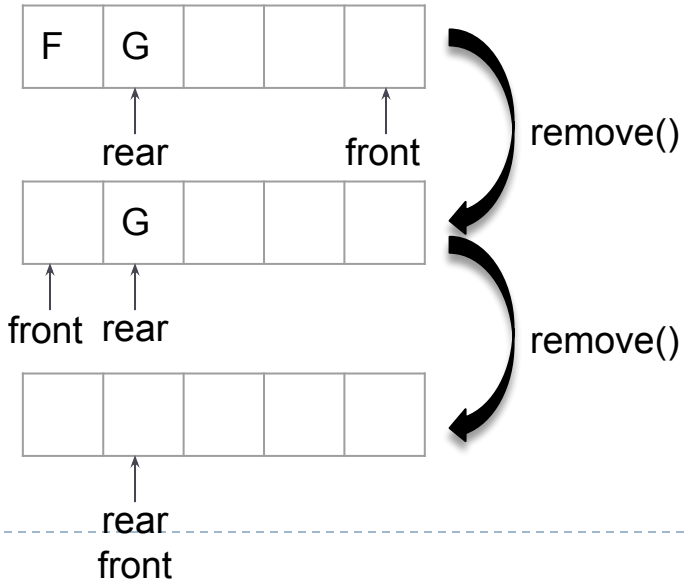
Problema???

- Visualizar o vetor como uma lista circular!



Problema???

- Quando a fila está vazia?? rear < front nao funciona mais
  - Pode atribuir q.front como índice do elemento anterior ao primeiro elemento, em vez de o próprio elemento
  - Se q.front==q.rear, a fila está vazia



Implementação

```
#define MAXQUEUE 100
struct queue {
  int items [MAXQUEUE];
    int front, rear;
};
struct queue q;
q.front = q.rear = MAXQUEUE -1;
```

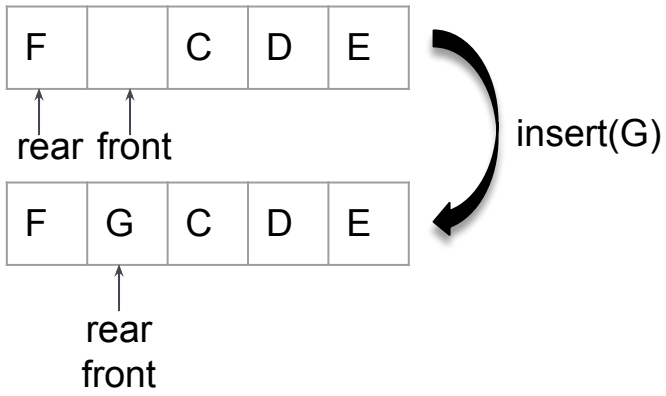
```c
int empty (struct queue *pq){
    if (pq->front==pq->rear)
        return 1;
    return 0;
}

int remove (struct queue *pq){
    if (empty(pq)){
        printf("underflow");
        exit(1);
    }

    //implementando a ideia de lista circular
    if (pq->front == MAXQUEUE -1)
        pq->front=0;
    else
        (pq->front)++;

    return pq->items[pq->front];
}
```

Verificação de estouro



```
void insert (struct queue *pq, int x){
    //implementa a ideia circular
    if (pq->rear == MAXQUEUE - 1)
        pq->rear=0;
    else
        (pq->rear)++;

    //verificação de estouro
    if (pq->rear==pq->front){
        printf("estouro de memória");
        exit(1);
    }

    pq->items[pq->rear]=x;
}
```

Problema???

- O teste de estouro de memória, no insert, acontece após rear ser atualizado
- O teste de underflow, no remove, ocorre antes da atualização de front
- Por quê??

Exercício

Implemente a lista circular que representa pessoas em uma fila de banco. Desta forma, cada elemento da lista tem a seguinte estrutura:

```
struct pessoa {
    char[11] cpf;
    char[100] nome;
    int senha;
    int idade;
}
```

Ao remover, mostre a senha da pessoa removida, dentro da rotina main, não dentro da função remove.