Curso: Ciência da Computação

Disciplina: Estruturas de Dados

Profª Luciana Ap. Oliveira Betetto

Aula 04 - Pilhas e Filas

1. Introdução

As pilhas e filas são conjuntos dinâmicos nos quais o elemento removido do conjunto pela operação **DELETE** é especificado previamente. Em uma ***pilha***, o elemento eliminado do conjunto é o mais recentemente inserido: a pilha implementa uma norma de ***último a entrar, primeiro a sair,*** ou ***LIFO*** (last-in, first-out). De modo semelhante, em uma ***fila***, o elemento eliminado é sempre o que esteve no conjunto pelo tempo mais longo: a fila implementa uma norma de ***primeiro a entrar, primeiro a sair,*** ou ***FIFO*** (first-in, first-out).

2. Estruturas de Dados do tipo Pilha e Fila

Nos estudos das **listas** percebe-se que o acesso era feito sem nenhuma restrição. Podia-se inserir e remover elementos em qualquer posição, só dependendo se a lista era ordenada ou não.

As Pilhas e as Filas são listas que possuem critérios de acesso restritivos:
LIFO - (Last Input, First Output) ou seja, o último a entrar numa lista é o primeiro a sair dela.
FIFO - (First Input, First Output) ou seja, o primeiro a entrar é o primeiro a sair.

Considerando-se os critérios de acesso, informalmente, pode-se identificar que uma PILHA é uma ***lista*** que tem como critério de acesso de acesso o ***LIFO***, e a ***fila***, o acesso ***FIFO.***

As estruturas de dados do tipo **pilha** e **fila** são consideradas **listas especializadas** por possuírem características próprias, mas também possuem operações como: inserir um elemento, excluir um elemento; encontrar o maior e o menor; contar os elementos, alterar e buscar um elemento, buscar o sucessor e o predecessor.

Essas duas estruturas de dados representam conjuntos de dados que estão organizados em ordem linear. Quando essas estruturas são representadas por arranjos, ou seja, quando é feita a utilização de vetores nas representações, tem-se o uso de endereços contíguos de memória do computador e a ordem linear é determinada pelos índices dos vetores, o que em algumas situações exige maior esforço computacional. Tais representações denominam-se pilha e fila estáticas. Quando as estruturas pilha e fila são representadas por elementos que, além de conter o dado, possuem também um ponteiro para o próximo elemento, ou seja, elementos encadeados, têm-se representações denominadas pilha e fila dinâmicas.

Quando um elemento de uma pilha ou de uma fila contem apenas um dado primitivo, como exemplo um número, chama-se pilha ou fila homogênea. Quando um elemento de uma pilha ou de uma fila mantem um dado composto, como o nome e o salário de um funcionário, chama-se pilha ou fila heterogênea.

2.1. Pilha ou LIFO (Last In First Out)

Uma Pilha é um conjunto de elementos no qual novos itens podem ser inseridos ou removidos em uma única extremidade, a qual é chamada de **TOPO**. A pilha possuirá características próprias de acesso e por consequência terá operações bem simples de empilhar e desempilhar elementos.

A operação **INSERT** sobre uma pilha é chamada com frequência de **PUSH**, e a operação **DELETE**, é frequentemente chamada de **POP**. Esses nomes são alusões a pilhas físicas, como as pilhas de pratos usados em restaurantes. A ordem em que os pratos são retirados da pilha é o oposto da ordem em que eles são colocados sobre a pilha e, como consequência, apenas o prato do topo está acessível.

Como mostra a Figura 1, podemos implementar uma pilha de no máximo ***n*** elementos com um arranjo S[1 .. ***n***]. O arranjo tem um atributo ***topo [S]*** que realiza a indexação do elemento inserido mais recentemente. A pilha consiste nos elementos ***S[1 .. topo[S]],*** onde ***S[1]*** é o elemento na parte inferior da pilha e ***S[topo[S]]*** é o elemento na parte superior (ou no topo).

Quando ***topo [S]*** = ***O***, a pilha contém nenhum elemento e está ***vazia***. É possível testar se a pilha está vazia, através da operação de consulta STACK-EMPTY. Se uma pilha vazia sofre uma operação de extração, dizemos que a pilha tem um ***estouro negativo***, que é normalmente um erro. Se ***topo [S]*** excede ***n***, a pilha tem um estouro positivo.

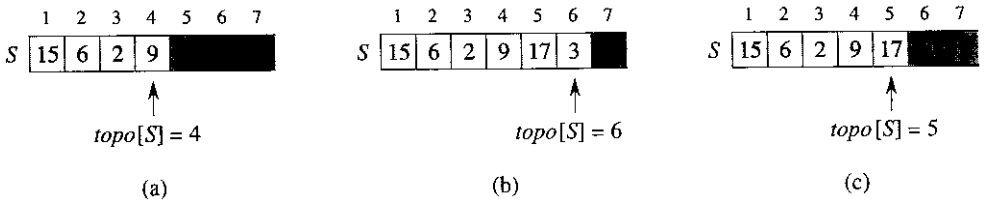


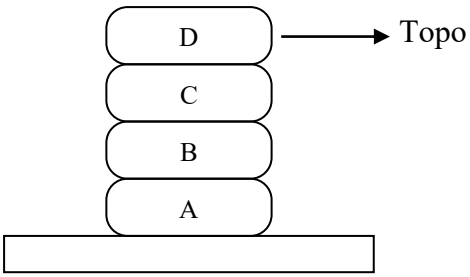
Figura 1. Uma implementação de arranjo de uma pilha S. Os elementos da pilha só aparecem nas posições levemente sombreadas. (a) A pilha S tem 4 elementos. O elemento do topo é 9. (b) A pilha S após as chamadas PUSH(S, 17) e PUSH(S, 3). (c) A pilha S após a chamada POP(S) retomou o elemento 3, que é o elemento mais recentemente inserido na pilha. O elemento do topo é o elemento 17.

Cada uma das operações sobre pilhas pode ser implementada com algumas linhas de código.

```
STACK-EMPTY(S)
if topo[S] = 0
        then return TRUE
        else return FALSE

PUSH(S, x)  // inserção
        topo[S] ← topo[S] + 1
        S[topo[S]] ← x

POP(S)  // remoção
if STACK-EMPTY(S)
        then error "underflow"
        else topo[S] ← topo[S] - 1
                return S[topo[S] + 1]
```

A Figura 2 mostra os efeitos das operações de modificação **PUSH** (EMPILHAR) e **POP** (DESEMPILHAR). Cada uma das três operações sobre pilhas demora o tempo O(1).

Veja a figura a seguir que possui quatro peças empilhadas rotuladas como A, B, C e D. Para a pilha abaixo ficar com essa configuração deve-se colocar primeiro a peça A, depois a B, C e D. No caso de retirada, só conseguiremos retirar na ordem: D, C, B e A.



A manipulação de uma pilha pode ser feita de forma **estática** ou **dinâmica**. A forma estática usará variável do tipo **vetor** e as pilhas dinâmicas usarão **ponteiros**. Numa pilha estática há preocupação com o tamanho da pilha, já que a alocação da variável é feita de forma estática. Assim, ao inserir um elemento tem-se de verificar se a pilha comporta. Na pilha dinâmica (que usa ponteiros) ao inserir um elemento tem-se de verificar se existe memória para a criação de um novo elemento na pilha.

Operações Associadas:

1. criar (P) - criar uma pilha P vazia
2. inserir (x, P) - insere x no topo de P (empilha)
3. vazia (P) - testa se P está vazia
4. topo (P) - acessa o elemento do topo da pilha (sem eliminar)
5. elimina (P) - elimina o elemento do topo de P (desempilha)

Um exemplo prático de Pilha pode ser, por exemplo, a lista de "últimas chamadas" (feitas ou recebidas) na memória de um telefone celular. Sempre a última chamada é a que aparece ao consultarmos esta lista, ou seja, o último valor a entrar é o primeiro a ser consultado (sair). E assim sucessivamente.

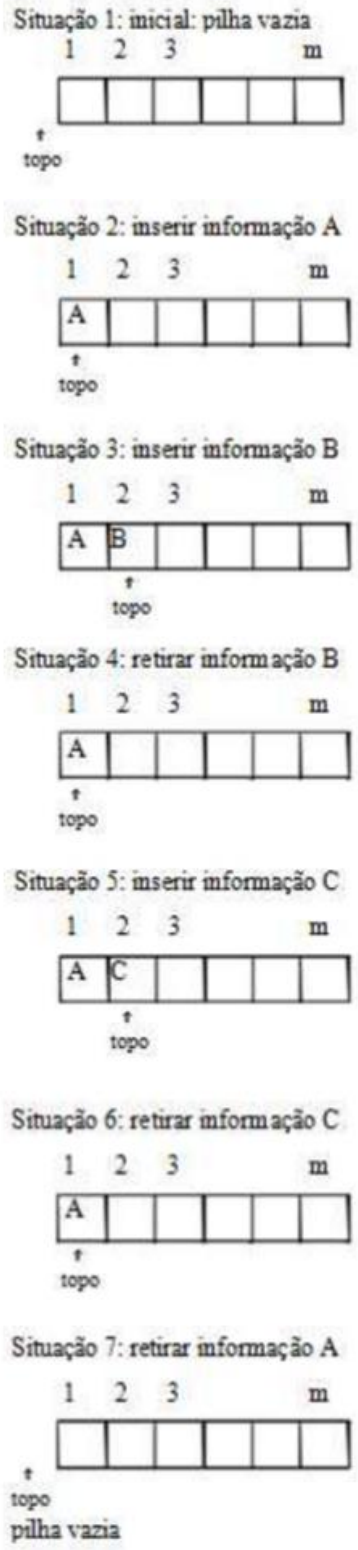
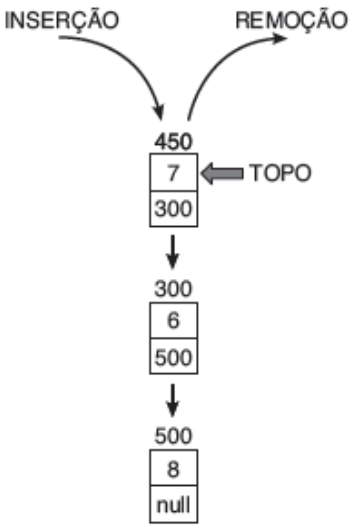
Operações: inserir na pilha, consultar toda a pilha, remover e esvaziá-la. Qualquer estrutura desse tipo possui um ponteiro denominado **TOPO**, no qual todas as operações de inserção e remoção acontecem. Assim, **as operações ocorrem sempre na mesma extremidade.**

Análise da complexidade: A ação de inserção e remoção na pilha sempre realiza operações básicas para atualizar o topo da pilha. São operações de tempo constante e gastam *b* A operação de consultar toda a pilha percorre os elementos armazenados nela. Considerando que uma pilha contém *n* elementos, logo o tempo de execução é *O(n).*

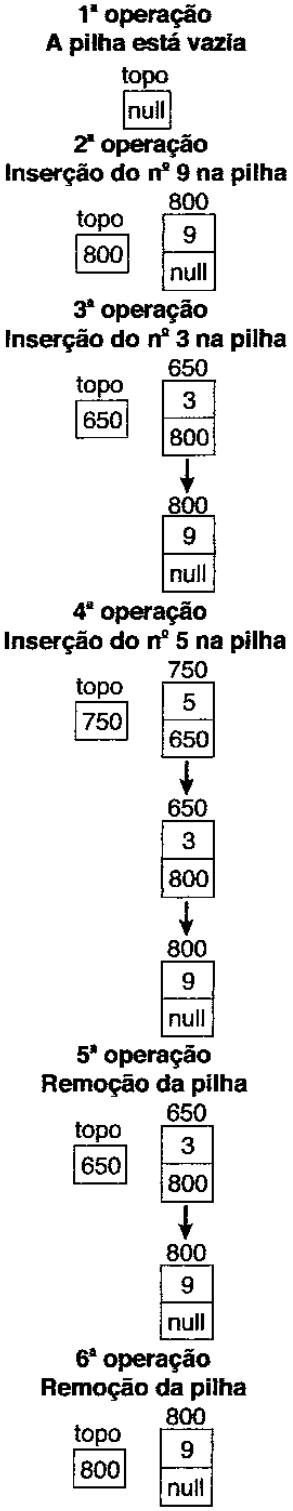
A operação de esvaziamento da pilha remove todos os elementos. O tempo gasto depende da linguagem de programação utilizada. No caso da JAVA, não é necessário remover cada um dos elementos, apenas atualiza-se o ponteiro topo da pilha para nulo e as memórias alocadas serão desalocadas por um procedimento JAVA. Já na linguagem C/C++, é necessário desalocar cada um dos elementos da pilha, gastando tempo proporcional ao tamanho dela, ou seja $O(n)$.

Implementação de Pilhas

Figura 2 - As pilhas podem ser implementadas usando *vetores* ou *ponteiros*.

Exemplo de pilha usando vetor:	***Exemplo de pilha usando ponteiro*** ***(os endereços de memória são ilustrativos):***
	

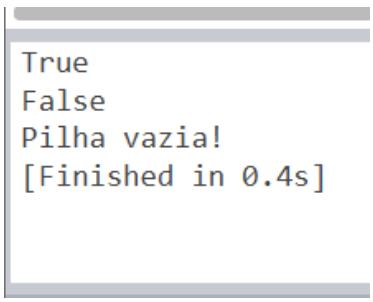
Exemplo de pilha usando ponteiro (os endereços de memória são ilustrativos):



Algoritmo para manipular uma PILHA:

Linguagem Python:

```
class Stack: # Pilha

	def __init__(self): #implementa a pilha utilizando a lista do python
		self.stack = [] # inicialmente, a pilha é vazia
		# criação da variável len_stack que aponta para o topo
		self.len_stack = 0 # se a pilha está vazia, seu comprimento é zero

	def push(self, e): # push insere elemento "e" na pilha
		self.stack.append(e) # o método append faz inserção no final
		self.len_stack += 1 # incrementa o comprimento da pilha

	def pop(self): # pop remove da pilha. Não precisa passar parâmetros
			# pq sempre remove do final
		if not self.empty(): # se a pilha não estiver vazia, remove
			self.stack.pop(self.len_stack - 1) # remove o último elemento
			self.len_stack -=1 # decrementa o comprimento da pilha

	def top(self): # top exibe o elemento do topo
		if not self.empty(): # se a pilha não estiver vazia
			return self.stack[-1]
		return('Pilha vazia!')

	def empty(self): # verifica se a pilha está vazia
		if self.len_stack == 0: # se comprimento da pilha for igual a zero
			return True
		return False

	def length(self): # retorna o tamanho da pilha
		return self.len_stack


TESTES:
s = Stack()
print(s.empty()) # verifica se a pilha está vazia, resultado 'True'
s.push(1) # insere o elemento 1
s.push(2) # insere o elemento 2
s.push(3) # insere o elemento 3
print(s.empty()) # verifica se a pilha está vazia, resultado 'False'
s.pop() # remove o elemento do topo, no caso, o elemento 3
s.pop() # remove o elemento do topo, no caso, o elemento 2
print(s.top()) # exibe o elemento no topo
s.pop() # remove o elemento do topo, no caso, o elemento 1
print(s.top()) # imprime 'Pilha vazia!'
```



Linguagem C:

```
#include <stdio.h>
//Definindo o registro que representará cada elemento da pilha
struct PILHA
{
	int num;
	PILHA *prox;
};

main()
{ //a pilha está vazia, logo, o ponteiro tem o valor NULL
  //as operações de inserção e remoção acontecem no TOPO
PILHA *topo = NULL;
//o ponteiro aux é um ponteiro auxiliar
PILHA *aux;
//apresentando o menu de opções
int op;
{
	printf (" MENU DE OPCOES:\n ");
	printf (" 1 - Inserir na pilha\n ");
	printf (" 2 - Consultar a pilha\n ");
	printf (" 3 - Remover da pilha\n ");
	printf (" 4 - Esvaziar a pilha\n ");
	printf (" 5 - Sair\n ");
	printf (" Digite sua opcao: ");
	scanf("%d",&op);
	if (op < 1 || op > 5)
		printf (" Opcao invalida !!\n ");

while (op == 1)
{
	printf (" Digite o número a ser inserido na pilha: ");
	PILHA *novo = new PILHA();  // criação da pilha
	scanf("%d",&novo->num);   // recebe o número que será armazenado na pilha
	novo->prox = topo;
	topo = novo;   //o número inserido está no topo
	printf (" Numero inserido na pilha !! \n");
	printf (" Digite sua opcao: ");
	scanf("%d",&op);
}

while (op == 2)
{
	if (topo == NULL)   //se o topo estiver NULO
	{
		//a pilha está vazia
		printf (" Pilha vazia ");
	}
	else
	{
		//a pilha contém elementos e estes serão
		//mostrados do último inserido ao primeiro
		printf (" Consultando a pilha: ");
		aux = topo;   //o ponteiro auxiliar aponta para o topo
		while (aux != NULL)   //enquanto o ponteiro aux for diferente de NULL
		{
```

```
                    printf ("num = %d\n" , aux->num);   //exibe o número
                    aux  =  aux->prox;     //o  ponteiro  aux  aponta  p/  o  próximo
elemento
             }
             printf (" Digite sua opcao: ");
             scanf("%d",&op);
      }
}
while (op == 3)
{
      if (topo != NULL)
      { //a pilha contém elementos e o último elemento
       //inserido será removido
       aux = topo;   //o ponteiro auxiliar aponta para o topo
       printf (" Numero removido: %d\n",topo->num);   //o número removido está no
topo
       topo = topo->prox;   //o topo aponta p/ prox (elemento abaixo do topo)
       delete(aux);   //deletar o elemento do topo
       printf (" Digite sua opcao: ");
       scanf("%d",&op);
      }

      else

      { //a pilha esta vazia
             printf (" Pilha vazia !! ");
             printf (" Digite sua opcao: ");
             scanf("%d",&op);
      }
}

while (op == 4)
{
      if (topo ==NULL)
      { //a pilha está vazia
             printf (" Pilha vazia !! ");
             printf (" Digite sua opcao: ");
             scanf("%d",&op);
      }
      else
      {
             //a pilha será esvaziada
             aux=topo;   //o ponteiro aux aponta p/ o topo
             while (aux!=NULL)   //enquanto o ponteiro aux for diferente de NULL
             {
                    topo = topo->prox;   // o topo aponta p/ o próximo
                    delete (aux);   //deletar o aux
                    aux=topo;   //aux recebe o número que está no topo
             }
             printf (" Pilha esvaziada !! ");
             printf (" Digite sua opcao: ");
             scanf("%d",&op);
      }
}

while (op !=5);

}
}
```

2.2. Fila ou FIFO (First In First Out)

Filas também são um caso especial de lista linear, nessa estrutura a regra a ser obedecida também é a ordem de inserção retirada dos elementos. Nessa estrutura, a preocupação também é somente com a ordem de entrada e saída dos mesmos, porém, nas filas deve ser feita somente nos dois extremos, **início** e **fim**.

Chamamos a operação **INSERT** sobre uma fila de **ENQUEUE** (ENFILEIRAR), e também a operação **DELETE** de **DEQUEUE** (DESINFILEIRAR); como a operação sobre pilhas POP, DEQUEUE não tem nenhum argumento de elemento. A propriedade FIFO de uma fila faz com que ela opere como uma fileira de pessoas no posto de atendimento da previdência social. A fila tem um ***início*** (ou cabeça) e um ***fim*** (ou cauda). Quando um elemento é colocado na fila, ele ocupa seu lugar no fim da fila, como um aluno recém-chegado que ocupa um lugar no final da fileira. O elemento retirado da fila é sempre aquele que está no início da fila, como o aluno que se encontra no começo da fileira e que esperou por mais tempo. (Felizmente, não temos de nos preocupar com a possibilidade de elementos computacionais "furarem" a fila.)

A Figura 3 mostra um modo de implementar uma fila de no máximo ***n - 1*** elementos usando um arranjo ***Q[1 .. n].*** A fila tem um atributo ***início[Q]*** que indexa ou aponta para seu início. O atributo ***fim[Q]*** realiza a indexação da próxima posição na qual um elemento recém-chegado será inserido na fila. Os elementos na fila estão nas posições ***início[Q], início[Q]*** + ***1, ... ,fim [Q] - 1***, onde "retornamos" , no sentido de que a posição 1 segue imediatamente a posição ***n*** em uma ordem circular. Quando ***início[Q]*** = ***fim[Q],*** a fila está vazia. Inicialmente, temos ***início[Q]*** = ***fim[Q]*** = ***1***. Quando a fila está vazia, uma tentativa de retirar um elemento da fila provoca o estouro negativo da fila. Quando ***início[Q]*** = ***fim[Q]*** + ***1***, a fila está cheia, e uma tentativa de colocar um elemento na fila provoca o estouro positivo da fila.

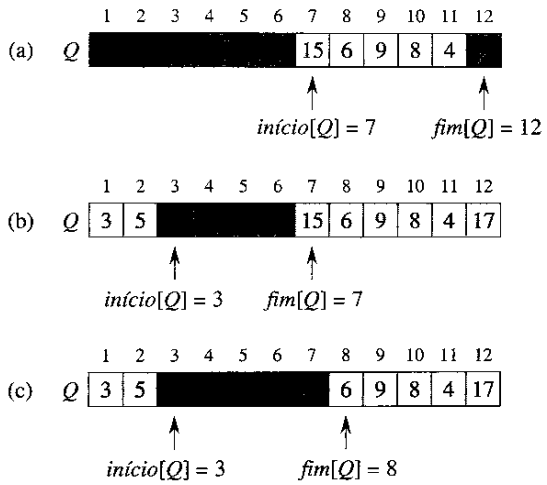


FIGURA 3. Uma fila implementada com a utilização de um arranjo Q[1..12]. Os elementos da fila aparecem apenas nas posições levemente sombreadas. (a) A fila tem 5 elementos, nas localizações Q [7 .. 11]. (b) A configuração da fila depois das chamadas ENQUEUE(Q, 17), ENQUEUE(Q, 3) e ENQUEUE(Q, 5). (c) A configuração da fila depois da chamada DEQUEUE(Q) retorna o valor de chave 15 que se encontrava anteriormente no início da fila. O novo início tem a chave 6

A Figura 3 mostra os efeitos das operações ENQUEUE e DEQUEUE. Cada operação demora o tempo O(1).

Em nossos procedimentos ENQUEUE e DEQUEUE, a verificação de erros de estouro negativo (underflow) e estouro positivo (overflow) foi omitida.

```
ENQUEUE(Q, x)  //enfileirar
Q[fim[Q]] ← x
        if fim[Q] = comprimento [Q]
                then fim[Q] ← 1
        else
                fim[Q] ← fim[Q] + 1

DEQUEUE(Q)  //desenfileirar
x ← Q[início[Q]]
        if início [Q] = comprimento [Q]
                then início [Q] ← 1
        else
                início[Q] ← início[Q] ← início[Q] + 1
                return x
```

A estrutura denominada **fila** é considerada do tipo **FIFO** (*First In First Out*), ou seja, o primeiro elemento inserido será o primeiro a ser removido. Nessa estrutura, cada elemento armazena um ou vários dados (estrutura homogênea ou heterogênea, respectivamente) e um ponteiro para o próximo elemento, permitindo o encadeamento e mantendo a estrutura linear.

Operações: inserir na fila, consultar toda a fila, remover e esvaziá-la. Essa estrutura possui um ponteiro denominado INÍCIO, onde as remoções acontecem, e um denominado FIM, onde as inserções acontecem. Assim, as operações ocorrem nas duas extremidades da estrutura.

Análise da complexidade: A operação de inserção na fila sempre realiza operações básicas, para atualizar o INÍCIO e FIM da fila. O mesmo ocorre no caso da remoção para atualizar o INÍCIO da fila. São operações de tempo constante e gastam tempo $O(1)$. A operação de consultar toda a fila percorre todos os elementos armazenados nela. Considerando que uma fila contém n elementos, logo o tempo de execução será $O(n)$.

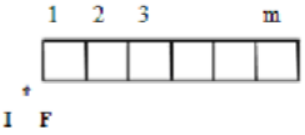
A operação de esvaziamento da fila consiste em remover todos os seus elementos. O tempo gasto depende da linguagem de programação utilizada. No caso da JAVA, não é necessário remover cada um dos elementos, apenas atualiza-se o ponteiro início da fila para nulo e as memórias alocadas serão desalocadas por um procedimento JAVA. Já na linguagem C/C++, é necessário desalocar cada um dos elementos da fila, gastando tempo proporcional ao tamanho dela, ou seja $O(n)$.

Implementação de Filas

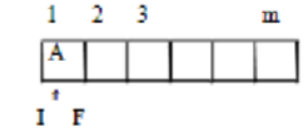
Figura 3 - As filas podem ser implementadas usando *vetores* ou *ponteiros*.

Exemplo de fila usando vetor:

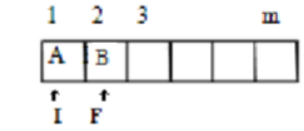
Situação 1: inicial: fila vazia



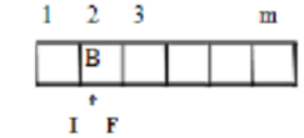
Situação 2: inserir informação A



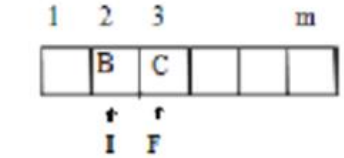
Situação 3: inserir informação B



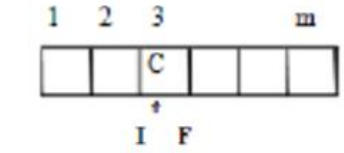
Situação 4: retirar informação A



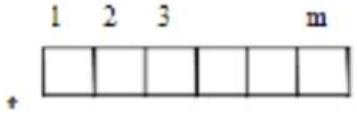
Situação 5: inserir informação C



Situação 6: retirar informação B



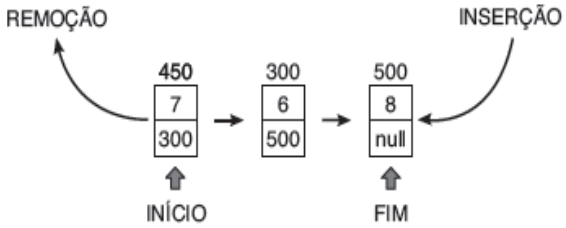
Situação 7: retirar informação C



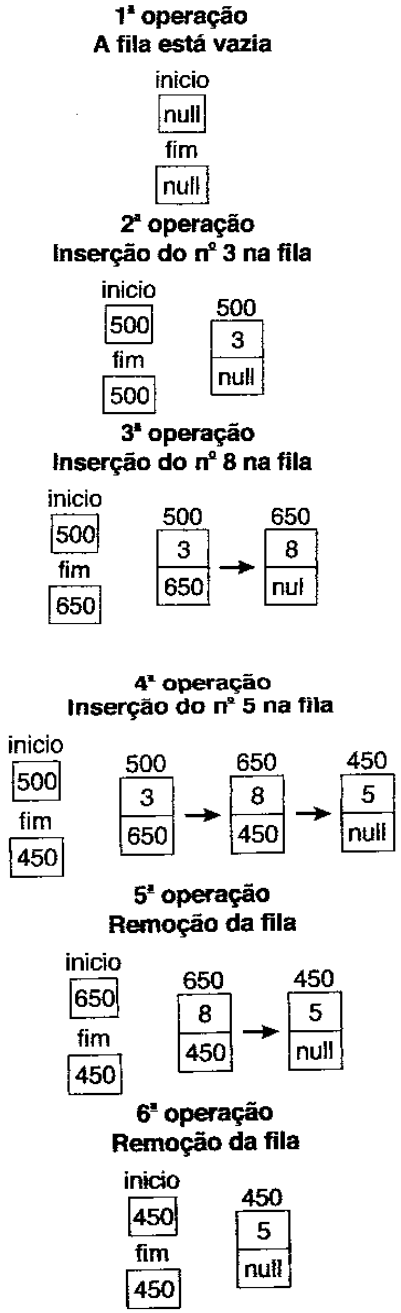
fila vazia

Exemplo de fila usando ponteiro

(os endereços de memória são ilustrativos):



Exemplo de fila usando ponteiro (os endereços de memória são ilustrativos):



Algoritmo para manipular uma FILA:

Linguagem Python:

```
class Queue:

	def __init__(self):
		self.queue = [] # no início a fila está vazia
		self.len_queue = 0 # essa variável indica o comprimento da fila
				   # se a fila está vazia, seu comprimento é zero

	def push(self, e): # insere um elemento na fila, sempre no final
		self.queue.append(e) # o método append faz inserção no final
		self.len_queue += 1 # incrementa o comprimento da fila

	def pop(self): # pop remove da fila, sempre do início
		if not self.empty(): # se a pilha não estiver vazia, então remove
			self.queue.pop(0)
			self.len_queue -= 1

	def empty(self): # verifica se a fila está vazia
		if self.len_queue == 0: # se comprimento da fila for igual a zero
			return True # a fila está vazia
		return False

	def length(self): # retorna o tamanho da fila
		return self.len_queue

	def front(self): # retorna o primeiro elemento da fila
		if not self.empty(): # se a pilha não estiver vazia
			return self.queue[0]
		return('Fila vazia!')
```

TESTES:

```
q = Queue()
print(q.empty()) # imprime se a fila está vazia no início
q.push(1) # insere o elemento 1 na fila
print(q.front()) # imprime o elemento da frente da fila, no caso, o 1
q.push(2) # insere o elemento 2 na fila
print(q.front()) # imprime o elemento da frente da fila, continua sendo o 1
q.push(3) # insere o elemento 3 na fila
print(q.empty()) # imprime se a fila está vazia
q.pop() # retira o elemento da fila, no caso, o elemento 1
print(q.front()) # imprime o elemento da frente da fila, no caso, o elemento 2
q.pop() # retira o elemento da fila, no caso, o elemento 2
print(q.front()) # imprime o elemento da frente da fila, no caso, o elemento 3
q.pop() # retira o elemento da fila, no caso, o elemento 3
print(q.front()) # imprime o elemento da frente da fila, mas a fila está vazia.
Então, imprime 'Fila vazia!'
```

Linguagem C:

```
#include <stdio.h>
//Definindo o registro que representará cada elemento da fila

struct FILA
{
	int num;
	FILA *prox;
};

main()
{
//a fila está vazia, logo, o ponteiro inicio tem o valor NULL
//a operação de remoção acontece no INICIO
//e a operação de inserção acontece no FIM
FILA *inicio = NULL;
FILA *fim = NULL;

//o ponteiro aux é um ponteiro auxiliar
FILA *aux;

//apresentando o menu de opções
int op;
{
	printf (" MENU DE OPCOES:\n ");
	printf (" 1 - Inserir na fila\n ");
	printf (" 2 - Consultar a fila\n ");
	printf (" 3 - Remover da fila\n ");
	printf (" 4 - Esvaziar a fila\n ");
	printf (" 5 - Sair\n ");
	printf (" Digite sua opcao: ");
	scanf("%d",&op);
	if (op < 1 || op > 5)
		printf (" Opcao invalida !!\n ");

while (op == 1)
{
	printf (" Digite o número a ser inserido na fila: ");
	FILA *novo = new FILA();
	scanf("%d",&novo->num);
	novo->prox = NULL;
	if (inicio == NULL)
	{
		//a fila está vazia e o número inserido
		//será o primeiro e o último
		inicio = novo;
		fim = novo;
	}
	else
	{
		fim->prox = novo;
		fim = novo;
	}
	printf (" Numero inserido na fila !! \n");
	printf (" Digite sua opcao: ");
	scanf("%d",&op);
}
while (op == 2)
```

```
{
	if (inicio == NULL)
	{
		//a fila está vazia
		printf (" Fila vazia !! ");
	}
	else
	{
		//a fila contém elementos e estes serão
		//mostrados do primeiro inserido ao último
		printf (" Consultando a fila: ");
		aux = inicio;
		while (aux != NULL)
		{
			printf ("num = %d\n" , aux->num);
			aux = aux->prox;
		}
		printf (" Digite sua opcao: ");
		scanf("%d",&op);
	}
}

while (op == 3)
{
	if (inicio != NULL)
	{//a fila contém elementos e o primeiro elemento
	 //inserido será removido
	 aux = inicio;
	 printf (" Numero removido: %d\n",inicio->num);
	 inicio = inicio->prox;
	 delete(aux);
	 printf (" Digite sua opcao: ");
	 scanf("%d",&op);
	}

	else

	{//a fila está vazia
		printf (" Fila vazia !! ");
		printf (" Digite sua opcao: ");
		scanf("%d",&op);
	}
}

while (op == 4)
{
	if (inicio ==NULL)
	{//a fila está vazia
		printf (" Fila vazia !! ");
		printf (" Digite sua opcao: ");
		scanf("%d",&op);
	}
	else
	{
		//a fila será esvaziada
		aux=inicio;
		while (aux!=NULL)
		{
			inicio = inicio->prox;
```

```
                    delete (aux);
                    aux=inicio;
                }
                printf (" Fila esvaziada !! ");
                printf (" Digite sua opcao: ");
                scanf("%d",&op);
        }
}

while (op !=5);

}
}
```

Referências:
Ascencio, A.F.G. & Araujo, G.S. Estruturas de Dados. Editora Pearson, 2011.
Laureano, Marcos. Estrutura de Dados com Algoritmos e C. Editora Brasport, 2008.