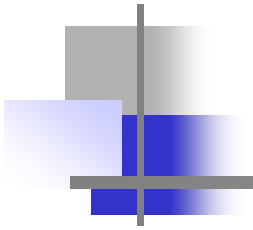


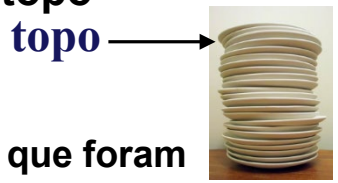
Programação II

Filas
(*Queues*)

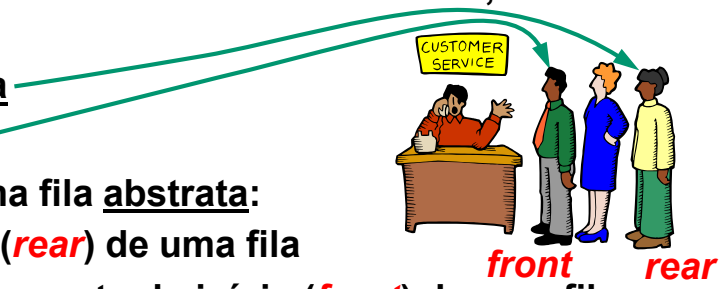
Bruno Feijó
Dept. de Informática, PUC-Rio

Fila

- **Na Pilha, o novo elemento é inserido no topo e o acesso é apenas ao topo**
  - **... como numa pilha de pratos**
  - **O único elemento que pode ser acessado e removido é o do topo**
  - **Na Pilha (*Stack*), os elementos são retirados na ordem inversa à ordem em que foram inseridos, *i.e.*: o último que entrou é o 1o. a sair (LIFO – *last in*, *first out*)**



- **Na estrutura Fila (*Queue*) é o inverso:**
  - **A analogia física é a fila de um banco. A ordem de uso é FIFO – *first in*, *first out* – *i.e.*: 1o. que entrou é o primeiro a sair**
  - **um novo elemento é inserido no final da fila**
  - **e um elemento é retirado do início da fila**



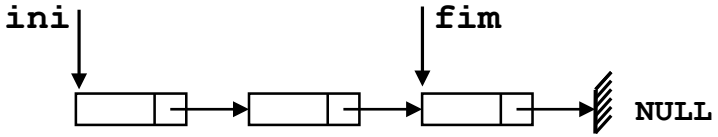
  - **Um conjunto mínimo de operações para uma fila abstrata:**
    - **`Inserir`: colocar um elemento no fim (*rear*) de uma fila**
    - **`Remover` (ou `Retirar`): remover um elemento do início (*front*) de uma fila e apresentá-lo**

      tentar remover elemento de uma lista vazia gera uma exceção.
    - **`Vazia`?: informar se a fila está ou não vazia**
  - **E, porque estamos implementando a fila em uma linguagem, ainda precisamos:**
    - **`Criar`: criar uma nova fila**
    - **`Liberar` (ou `Destruir`): destruir a fila quando não precisarmos mais dela**

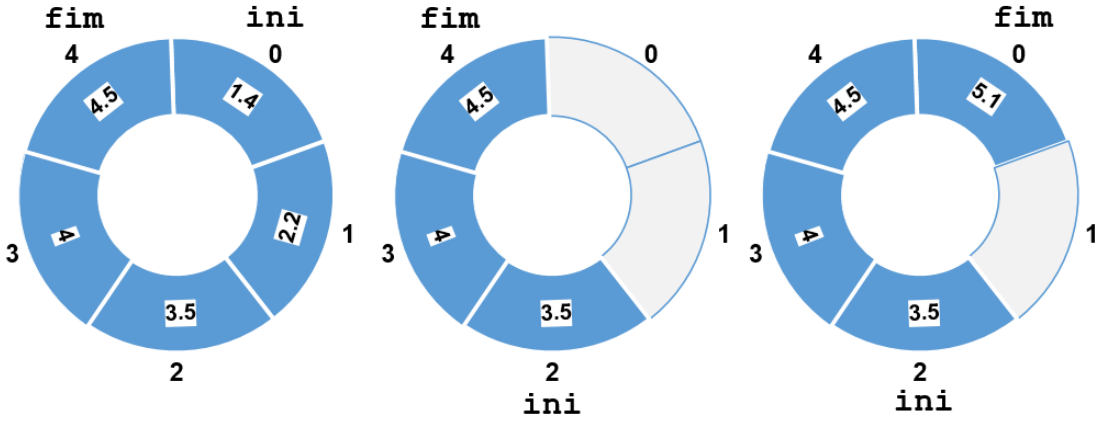
Abstração (*abstraction*) é um princípio muito poderoso em computação. Trata-se do ato de representar aspectos (*features*) essenciais de objetos computacionais sem precisarmos conhecer detalhes da sua organização física concreta e da possibilidade de usarmos estes objetos através de suas funcionalidades. Você viu um pouco disto em Pilhas (… e verá mais em Programação Modular e Orientação a Objetos).

Formas de Implementar Filas

- **Como uma lista encadeada:**



- **Como vetor, numa forma circular**
  - **Tamanho fixo N. Acesso rápido e pouca memória.**
  - **No exemplo abaixo (N = 5), há primeiro a retirada de dois elementos (1.4 e 2.2) e depois a inserção de um novo elemento (5.1)**



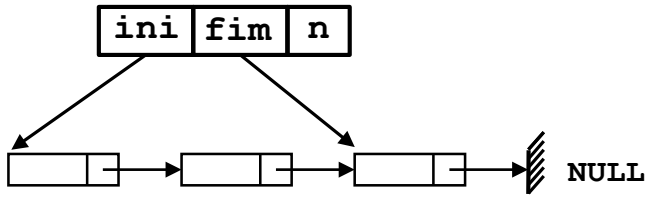
1.4	2.2	3.5	4	4.5
0	1	2	3	4
ini				fim

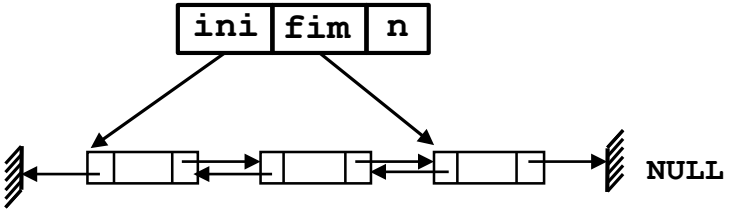
		3.5	4	4.5
0	1	2	3	4
		ini		fim

5.1		3.5	4	4.5
0	1	2	3	4
fim		ini		

Fila como Lista Encadeada

- É fácil ter tamanho livre. Porém requer mais memória e a memória é fragmentada.
- Podemos definir a fila como uma estrutura que tem os ponteiros `ini` e `fim` (e, eventualmente, o tamanho da fila, `n`). Pode ser simples ou duplamente encadeada





- Também pode ser <u>circular</u> simplesmente encadeada (Fig. a) ou <u>circular</u> duplamente encadeada (Fig. b). Usamos estes casos ao invés do vetor circular quando há a necessidade de alterar o tamanho da fila (pois alteração de tamanho como vetor requer realocação e cópia). A estrutura [`ini` , `fim` , `n`] também pode ser usada (Fig. c)

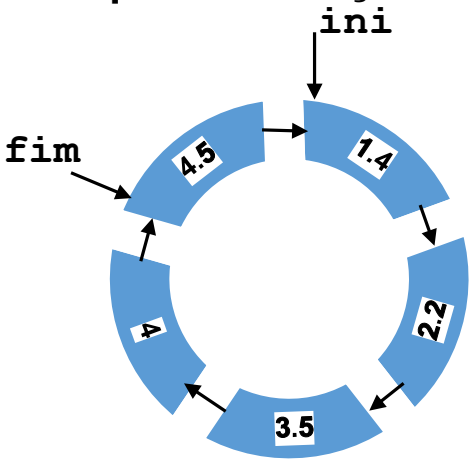


Fig. a

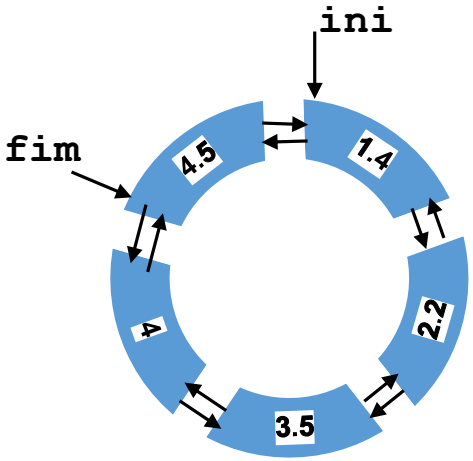


Fig. b

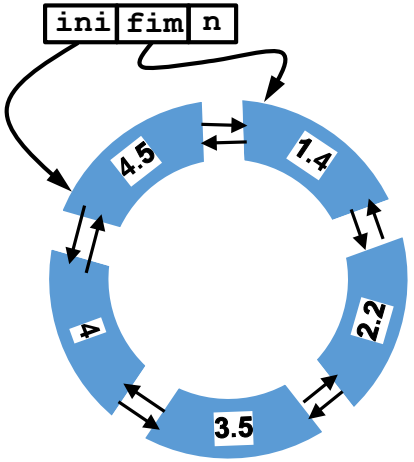


Fig. c

Fila de Prioridade (Priority Queue)

- **Priority Queue: fila onde cada elemento tem um atributo de prioridade e a função de remoção encontra o elemento com a prioridade mais alta**

Fila Dupla (*Deque*)

- **Fila Dupla (*Double Ended Queue*) é a fila na qual é possível:**
  - **inserir novos elementos no início e no fim**
  - **retirar elementos de ambos os extremos**

- **Se usar listas encadeadas: deve ser a duplamente encadeada (a simplesmente encadeada seria extremamente ineficiente)**
- **Também denominada pela abreviação *Deque* (pronunciamos "*deck*")**
- **Simula, dentro de uma mesma estrutura, duas filas, com os elementos em ordem inversa uma da outra. É uma generalização simultânea de pilha e fila.**
- **Ainda há variantes**
  - **Input Restricted Queue:**
    - **remoção pode ocorrer nas duas extremidades**
    - **Inserção apenas no fim (*rear*)**
  - **Output Restricted Queue:**
    - **remoção apenas no início (*front*)**
    - **Inserção pode ocorrer nas duas extremidades**
- **Uso: quando o tratamento do início de cada fila tem frequência diferente. Por exemplo, o OpenGL exibe imagens mais rapidamente do que o módulo que gera a imagem (*e.g.*, o módulo gerador para um pouco para *garbage collection*)**

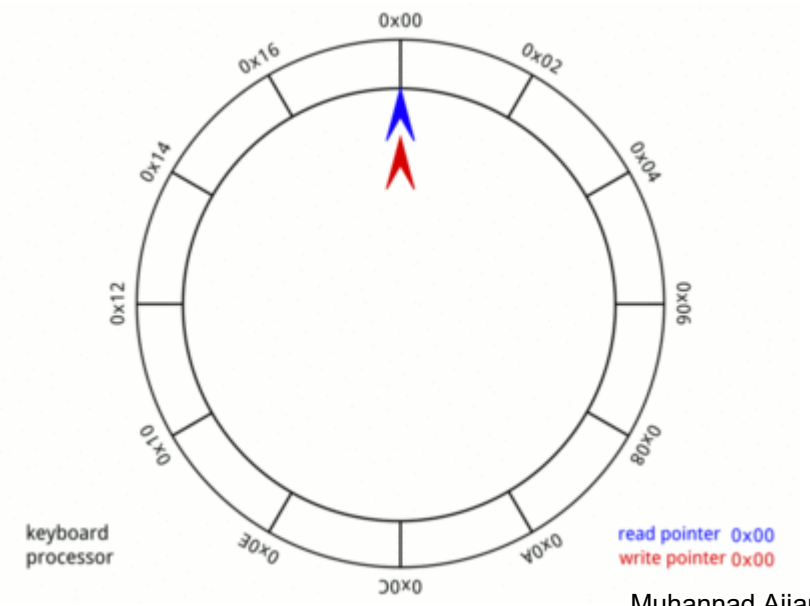
Fila Dupla (*Deque*)

- **Nomes de função para *Deque*:**

Operação	Nome da Função
Insere elemento no fim	`pushRear` ou `insereFim`
Insere elemento no início	`pushFront` ou `insereIni`
Remove último elemento	`popRear` ou `retiraFim`
Remove primeiro elemento	`popFront` ou `retiraIni`

Exemplos de Uso de Filas - Teclado

- **Pressionar teclas no teclado**
  - **Fila circular como vetor**
  - **Quando a escrita do caractere encher a fila (o sistema não aceita novos caracteres e emite um beep). A leitura do caracter pelo processador do teclado vai esvaziando a lista.**
  - **Também conhecido como Circular Buffer**

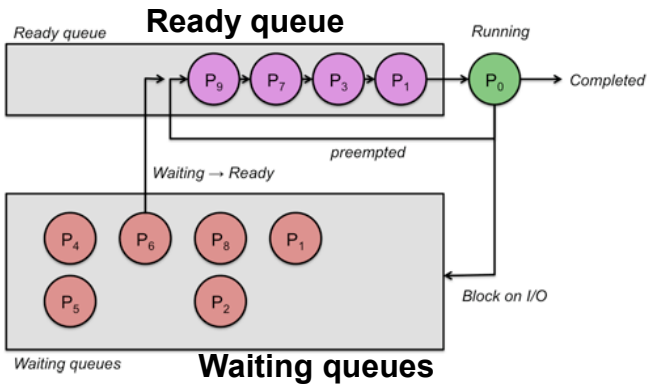


Muhannad Ajjan, 2015 (Wikimedia Commons)

Para ver a animação clique aqui

Exemplos de Uso de Filas – Agendamento *Round Robin*

- ***Round Robin Scheduling* para Gerenciamento de Processos**
  - **Usado em sistemas multi-usuários e de compartilhamento de tempo (*time-sharing*)**
  - **Isto poderia ser feito com *Priority Queue*, mas *Round Robin* é mais simples e com baixa sobrecarga na tomada de decisão. E também é um algoritmo que distribui o tempo de resposta mais uniformemente (chegando a um melhor tempo de resposta médio).**
  - **Usamos fila circular implementada como vetor.**
  - **"*Round Robin*" era o nome dado a um documento de petição assinado por várias pessoas na forma de um círculo para evitar a identificação de um eventual líder ou cabeça de lista (chamado na França de *ruban rond ou rond ruban*: tira circular, *round ribbon* em inglês).**



1. **Um mesmo tempo fixo t é designado para os processos que chegam na fila, chamado *time slice* ou *time quantum*. Desempenho depende de t e no. de processadores.**
2. **O primeiro processo que chega é selecionado e enviado para o processador para execução. Se o processador não é capaz de completar a sua execução dentro do *time quantum*, então uma interrupção é gerada.**
3. **O processo interrompido tem o estado salvo na memória (para ser possível retomar o processo a partir do ponto que foi interrompido) (infelizmente, há sempre *overhead* p/mudança de contexto). Neste momento, o processo é enviado de volta para o fim da fila, mas sempre depois de um novo processo que chega.**
4. **O *scheduler* seleciona um outro processo da fila e o despacha para o processador, e segue os mesmos passos acima.**

Especificação da interface do tipo *Fila*

- **Interface do tipo abstrato *Fila: fila.h***
  - **função fila_cria**
    - **aloca dinamicamente a estrutura da fila**
    - **inicializa seus campos e retorna seu ponteiro**
  - **função fila_libera**
    - **destrói a fila, liberando toda a memória usada pela estrutura**
  - **função fila_vazia**
    - **informa se a fila está ou não vazia**
  - **função fila_insere e função fila_retira**
    - **insere e retira, respectivamente, um valor na fila**
    - **retirar valor de uma fila vazia gera uma exceção**

Interfaces de Tipos Abstratos

- **Não vamos explorar tudo o que a idéia de um tipo abstrato pode trazer de vantagens. Por enquanto, recomendamos que você sempre defina tipos que são mais complexos (como uma *Fila*, por exemplo) através de suas interfaces (que especificam o tipo, através do *typedef*) e através de suas funcionalidades (*i.e.*, as funções de serviço, especificadas pelos protótipos).**
- **E se você quiser explorar ainda mais um pouco esta idéia de abstração, coloque a interface do tipo abstrato *Fila* no arquivo *fila.h***
- **No próximo slide, damos uma sugestão de interface de *Fila* (que não é a mais genérica possível quando o tipo da informação é um dado composto, mas que é a mais fácil de você entender no momento).**

Exemplo de interface do tipo *Fila*

Dado simples, e.g. um inteiro (**int**):

```
typedef struct fila Fila;

Fila * fila_cria (void);
void fila_libera(Fila * f);
int fila_vazia(Fila * f);
void fila_insere(Fila * f, int v);
int fila_retira(Fila * f);
```

Dado composto, e.g. uma struct (**Cliente**):

```
typedef struct cliente Cliente;
typedef struct fila Fila;

Fila * fila_cria (void);
void fila_libera(Fila * f);
int fila_vazia(Fila * f);
void fila_insere(Fila * f, Cliente * v);
Cliente * fila_retira(Fila * f);
```

Obs: se você quisesse ser mais genérico, você definiria um `tipoElemFila` **que poderia ser um float:**

```
typedef float tipoElemFila;
```

ou um ponteiro para Cliente:

```
typedef struct cliente * tipoElemFila;
```

e definiria, por exemplo, que:

```
void fila_insere(Fila * f, tipoElemFila v);
```

Mas, deixe isto para um curso mais avançado.

Vamos ver, agora, alguns tipos de implementação de filas

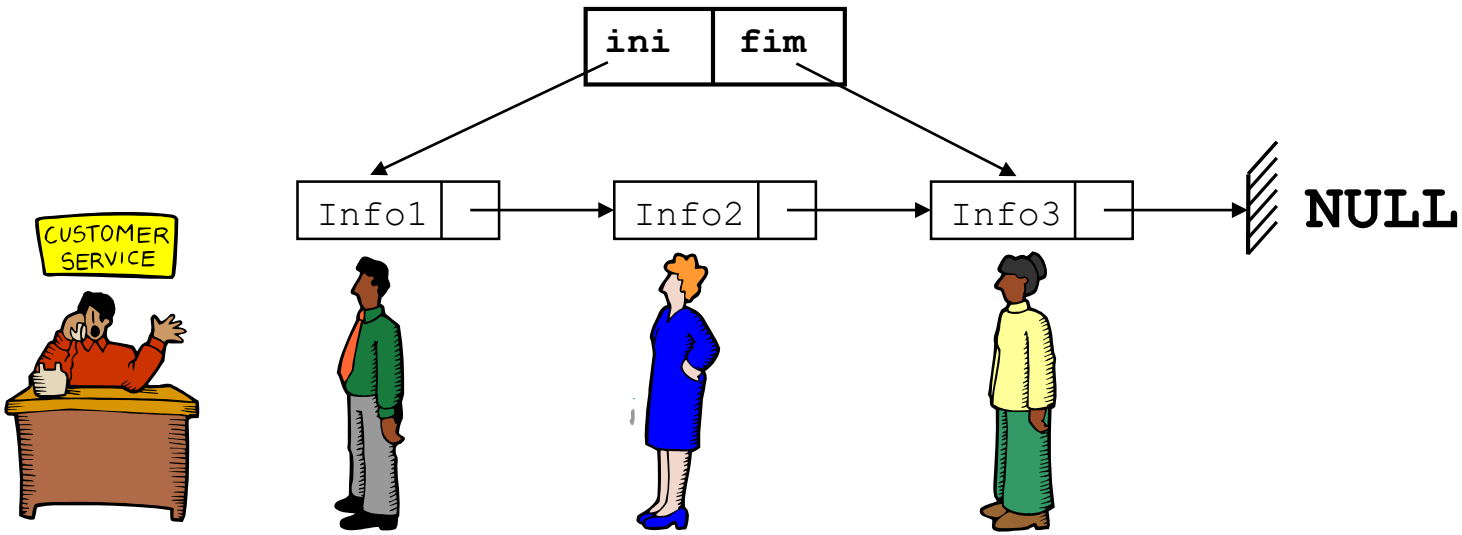
FILA COMO LISTA

Implementação de fila com lista

- **Implementação de fila com lista**
  - **elementos da fila armazenados na lista**
  - **usa dois ponteiros**

    ini (ou *front*) **aponta para o primeiro elemento da fila**

    fim (ou *rear*) **aponta para o último elemento da fila**



Implementação de fila com lista

- **Implementação de fila com lista**
  - elementos da fila armazenados na lista
  - fila representada por um ponteiro para o primeiro nó da lista

```
/* elemento de fila */
struct elemFila {
   tipo info;
   struct elemFila * prox;
};
typedef struct elemFila ElemFila;

/* estrutura da fila (tipo Concreto) */
struct fila {
   ElemFila * ini;  // ou: QueueNode * front;
   ElemFila * fim;  // ou: QueueNode * rear;
};
```

*pode ser float, int, Cliente **

Implementação de fila com lista

- Função *fila_cria*
  - *fila_cria* aloca a estrutura da fila (usuário é que testa se retorna NULL)
  - inicializa a lista como sendo vazia

```
Fila * fila_cria(void)
{
   Fila * f = (Fila *)malloc(sizeof(Fila));
   if (f != NULL)
      f->ini = f->fim = NULL;
   return f;
}
```

- Função fila_vazia

```
int fila_vazia(Fila * f)
{
   return (f->ini == NULL);
}
```

Implementação de fila com lista

- função fila_insere
  - insere novo elemento v no final da lista

*pode ser float, int, Cliente **

```
void fila_insere (Fila * f, tipo v)
{
   ElemFila * n = (ElemFila *)malloc(sizeof(ElemFila));
   assert(!(n == NULL));
   n->info = v;          /* armazena informacao */
   n->prox = NULL;       /* novo nó passa a ser o último */
   if (!fila_vazia(f)) /* verifica se lista não e' vazia */
      f->fim->prox = n;
   else                  /* fila estava vazia */
      f->ini = n;
   f->fim = n;           /* fila aponta para novo elemento */
}
```

ou

```
if (n==NULL) {printf("sem memoria\n"); exit(1);}
```

assert() requer #include <assert.h>

assert(*expressão*); se *expressão* **não é** verdade, a execução termina e mensagem de erro identifica arquivo, linha, função e condição que foi violada.

Implementação de fila com lista

- função fila_retira
  - retira o elemento do início da lista

```
tipo fila_retira(Fila * f)
{  ElemFila * t;
   tipo v;
   assert(!fila_vazia(f));
   t = f->ini;
   v = t->info;
   f->ini = t->prox;
   if (f->ini == NULL) /* verifica se fila ficou vazia */
      f->fim = NULL;
   free(t);
   return v;
}
```

*pode ser float, int, Cliente **

ou

```
if (fila_vazia(f))
   { printf("Fila vazia.\n"); exit(1);} /* aborta */
```

Implementação de fila com lista

- função fila_libera
  - libera a fila depois de liberar todos os elementos da lista

```
void fila_libera (Fila * f)
{
   ElemFila * q = f->ini;
   ElemFila * t;
   while (q!=NULL)
   {
      t = q->prox;
      free(q);
      q = t;
   }
   free(f);
}
```

FILA CIRCULAR COMO VETOR

Implementação de fila com vetor

- **Implementação de fila com vetor**
  - vetor (vet) armazena os elementos da fila
  - estrutura de fila:

```
#define N  100          /* numero máximo de elementos */

struct fila {
   int n;               /* numero de elementos na fila */
   int ini;
   tipo vet[N];
};      → pode ser float, int, Cliente *
```

  - A princípio, o `fim` seria calculado como `fim = ini + n - 1` (no exemplo abaixo à esquerda, `ini = 0`, `n = 5` e `fim = 4`). Entretanto, se retirarmos os dois primeiros elementos com um `fila_retira` e logo após inserirmos um novo elemento, ele poderia ser colocado na posição vaga `0` (e `fim` seria `0`). Precisamos trabalhar de uma <u>forma circular</u>!

1.4	2.2	3.5	4	4.5
0	1	2	3	4
ini				fim

		3.5	4	4.5
0	1	2	3	4
		ini		fim

5.1		3.5	4	4.5
0	1	2	3	4
fim		ini		

Implementação de fila com vetor

- **Cálculos para atualização dos índices de forma circular:**
  - **incremento das posições do vetor de forma "circular":**
    - **usamos o operador módulo "%" (é o resto: 5%3=2, 3%5=3, 1%5=1, 5%5=0)**
    - **Vamos usar a propriedade de que `i%N = i` (se `i<N`) e `i%N = 0` (se `i=N`)**
    - **Ou seja: quando o índice chegar à última posição, ele volta ao índice `0`.**
  - **Para calcular a posição atual de `fim`: `fim = (ini + n - 1)%N`**
  - **Para calcular a nova posição de `fim` onde será inserido um novo elemento (corte o `-1`): `fim = (ini + n)%N`**
  - **Se for retirar um elemento, a posição `ini` precisa ser atualizada para: `ini = (ini + 1)%N`**
  - **Faça a simulação: a partir de (a) insira novo elemento `5.1` (b), depois retire o elemento no início (`3.5`) e, por fim, insira `6.1` (c):**

		3.5	4	4.5
0	1	2	3	4
		ini		fim

(a)

5.1		3.5	4	4.5
0	1	2	3	4
fim		ini		

(b)

5.1	6.1		4	4.5
0	1	2	3	4
	fim		ini	

(c)

Implementação de fila com vetor

- função fila_cria
  - aloca dinamicamente um vetor
  - inicializa a fila como sendo vazia (número de elementos = 0)

```
Fila * fila_cria (void)
{
   Fila * f = (Fila *)malloc(sizeof(Fila));
   f->n = 0;       /* inicializa fila como vazia */
   f->ini = 0;     /* escolhe uma posição inicial */
   return f;
}
```

- Função fila_vazia

```
int fila_vazia (Fila * f)
{
   return f->n==0;
}
```

Agradecemos revisão de alguns detalhes deste slide ao aluno Vitor Krau (Turma 2017.2)

Implementação de fila com vetor

- função fila_insere
  - insere um elemento no final da fila
  - usa a próxima posição livre do vetor, se houver

```
void fila_insere(Fila * f, tipo v)
{   int fim;
    assert(!(f->n == N));
    /* insere elemento na proxima posicao livre */
    fim = (f->ini + f->n) % N;
    f->vet[fim] = v;
    f->n++;
}
```

ou

```
if (f->n == N) { /* fila cheia: capacidade esgotada */
      printf("Capacidade da fila estourou.\n");
      exit(1);       /* aborta programa */ }
```

Implementação de fila com vetor

- função fila_retira
  - retira o elemento do início da fila, retornando o seu valor
  - verifica se a fila está ou não vazia

```
tipo fila_retira(Fila* f)
{   tipo v;
    assert(!fila_vazia(f));
    /* retira elemento do início */
    v = f->vet[f->ini];
    f->ini = (f->ini + 1) % N;
    f->n--;
    return v;
}
```

ou

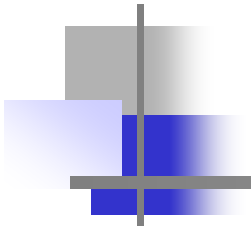
```
if (fila_vazia(f)) {
      printf("Fila vazia.\n");
      exit(1);        /* aborta programa */  }
```

Agradecemos revisão de alguns detalhes deste slide ao aluno Vitor Krau (Turma 2017.2)

Implementação de fila com vetor

- **função fila_imprime (se usamos tipo genérico, não é possível escrevê-la)**
  - **De qualquer maneira, esta função é difícil de ser escrita**
  - **Simplificação: usar especificamente para fila de tipos primitivos (*float*, *int*, ...)**

```
void fila_imprime(Fila * f)
{
   int i;
   int ini, fim, n;
   if (fila_vazia(f))
   {
      printf("Fila vazia\n");
      return;
   }
   ini = f->ini;
   n = f->n;
   fim = (ini + n - 1) % N;
   printf("Inicio[%d]->",ini);       // imprime o ini
   for (i=ini; i!=fim; i=(i+1)%N)
      printf("%7.2f  ", f->vet[i]);
   printf("%7.2f  ", f->vet[i]);     // imprime o ultimo
   printf("<-[%d]Fim\n",fim);        // imprime o fim
}
```



FILA DUPLA (*DEQUE*) COMO VETOR

Interface do tipo fila dupla

- **Interface do tipo abstrato Fila2*: fila2.h***
  - **função fila2_cria**
    - **aloca dinamicamente a estrutura da fila**
    - **inicializa seus campos e retorna seu ponteiro**
  - **função fila2_insere_fim e função fila2_retira_ini**
    - **insere no fim e retira do início, respectivamente, um valor real na fila**
  - **função fila2_insere_ini e função fila2_retira_fim**
    - **insere no início e retira do fim, respectivamente, um valor real na fila**
  - **função fila2_vazia**
    - **informa se a fila está ou não vazia**
  - **função fila2_libera**
    - **destrói a fila, liberando toda a memória usada pela estrutura**

Interface do tipo fila dupla

```
typedef struct fila2 Fila2;

Fila2* fila2_cria (void);

void fila2_libera (Fila2* f);

void fila2_insere_ini (Fila2* f, float v);

void fila2_insere_fim (Fila2* f, float v);

float fila2_retira_ini (Fila2* f);

float fila2_retira_fim (Fila2* f);

int fila2_vazia (Fila2* f);
```

Implementação de fila dupla com vetor

- **função fila2_insere_ini**
  - **insere elemento no início da fila**
    - **índice do elemento que precede ini é dado por (ini - 1 + N) % N**

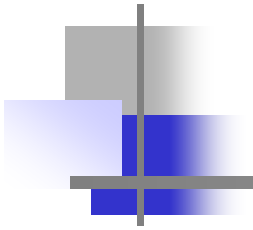
```
void fila2_insere_ini (Fila* f, float v)
{  int prec;
   if (f->n == N) {   /* fila cheia: capacidade esgotada */
      printf("Capacidade da fila estourou.\n");
      exit(1);        /* aborta programa  */
   }
   /* insere elemento na posição precedente ao início       */
   prec = (f->ini - 1 + N) % N;       /* decremento circular */
   f->vet[prec] = v;
   f->ini = prec;      /* atualiza índice para início     */
   f->n++;
}
```

ou use `assert(!(f->n == N));`

Implementação de fila dupla com vetor

- **função fila2_retira_final**
  - **retira elemento do final da fila**
    - **índice do último elemento é dado por (ini+n-1)%N**

```
float fila_retira (Fila* f)
{ float v;
   if (fila_vazia(f)) {
      printf("Fila vazia.\n");
      exit(1);        /* aborta programa */
   }
   /* retira elemento do início */
   v = f->vet[f->ini];
   f->ini = (f->ini + 1) % N;
   f->n--;
   return v;
}
```



Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel, *Introdução a Estruturas de Dados*, Editora Campus (2004)

Capítulo 12 – Filas