Pilhas e Filas

Estrutura de Dados

Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação

Conteúdo

Introdução

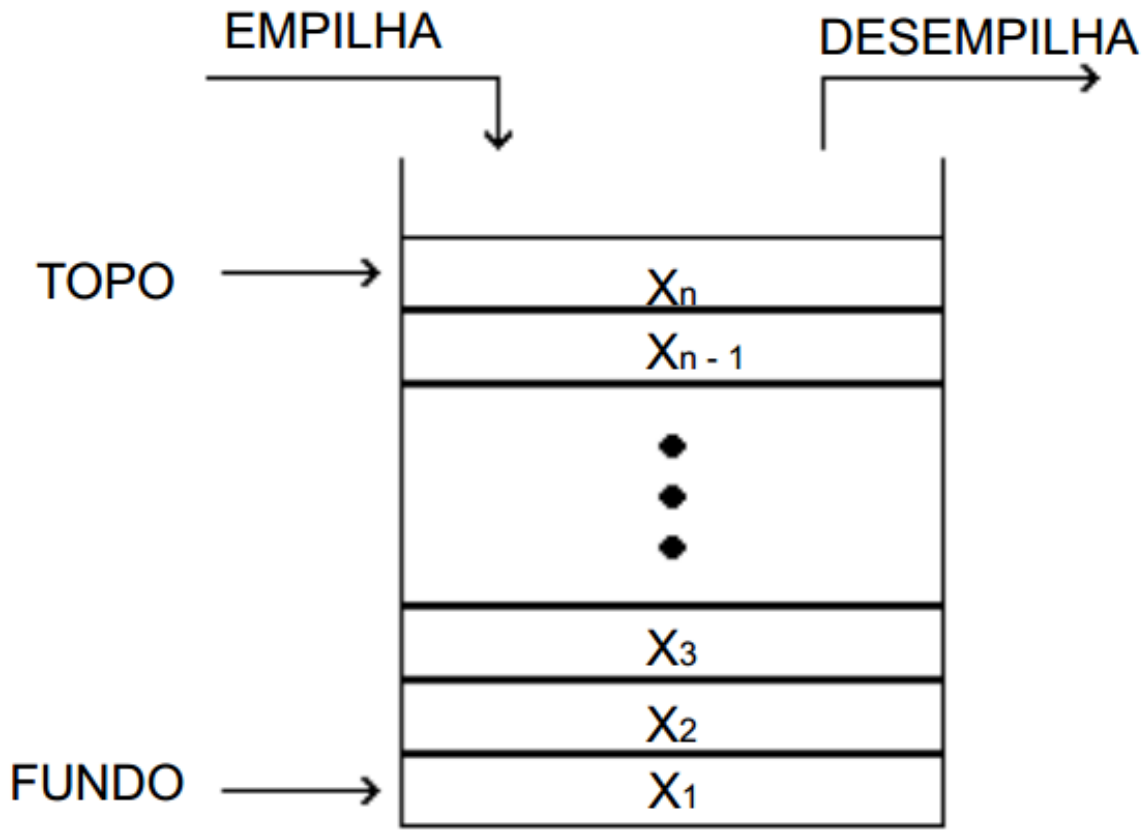
Pilhas e Filas

- Pilhas e Filas são estruturas de dados utilizadas em várias aplicações.
- Pilhas e Filas são **listas** que possuem uma **disciplina de acesso** bem definida.
- Exemplos:
  - Pilha de pratos
  - Fila de banco

Pilhas

- A pilha é uma lista onde todas as inclusões e exclusões de nós são feitas em uma **única extremidade**.
- Essa extremidade é conhecida como **topo**.
- Usa-se o critério de retirar em primeiro lugar o último nó que foi incluído na lista.
- Este critério é conhecido como **FILO** (**F**irst **I**n **L**ast **O**ut).
- A ideia desta estrutura é a de uma pilha de pratos.

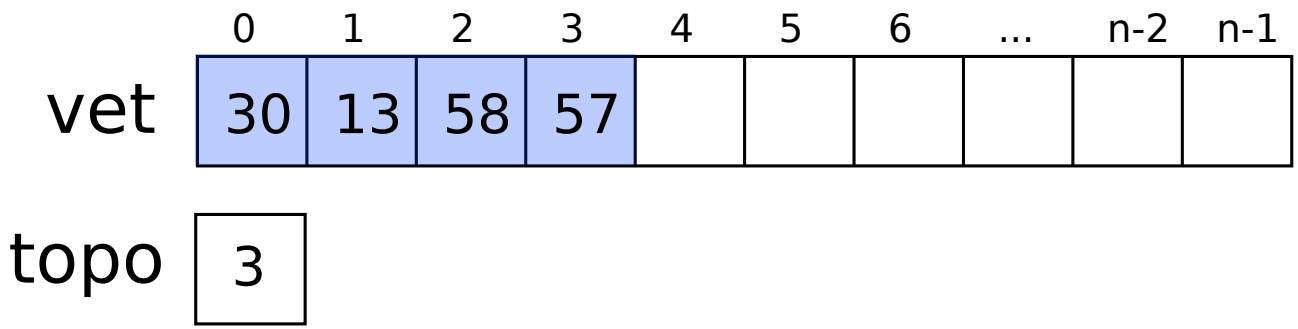
Pilhas



Pilhas

Representação com vetor

- Representação com vetor (ou contiguidade dos nós).
- Esquematicamente, uma pilha contígua com capacidade máxima para n nós pode ser representada da seguinte forma:



Pilhas

Representação com vetor

- Desta forma, o TAD `PilhaCont` para representar uma pilha de números inteiros, pode ser implementado como:

```
class PilhaCont
{
 private:
  int max;  // capacidade maxima
  int topo; // posicao do topo da pilha
  int *vet; // vetor que armazena os dados da pilha

 public:
   PilhaCont(int tam);
   ~PilhaCont();
   int getTopo();          // valor do No no Topo
   void empilha(int val);// insere No no Topo
   int desempilha();       // elimina No do Topo
   bool vazia();
};
```

Pilhas

Representação com vetor

```
PilhaCont::PilhaCont(int tam)
{
  max  = tam;
  topo = -1;
  vet = new int[max];
}

PilhaCont::~PilhaCont()
{
  delete [] vet;
}
```

- O construtor cria inicialmente uma pilha vazia.
- A pilha vazia é marcada com topo com o valor -1.
- Condição de pilha vazia: `topo == -1`.

Pilhas

Representação com vetor

```
int PilhaCont::getTopo()
{
  if(topo != -1)
    return vet[topo];
  else
  {
    cout << "ERRO: Pilha vazia!" << endl;
    exit(1);
  }
}

bool PilhaCont::vazia()
{
  return (topo == -1);
}
```

Pilhas

Representação com vetor

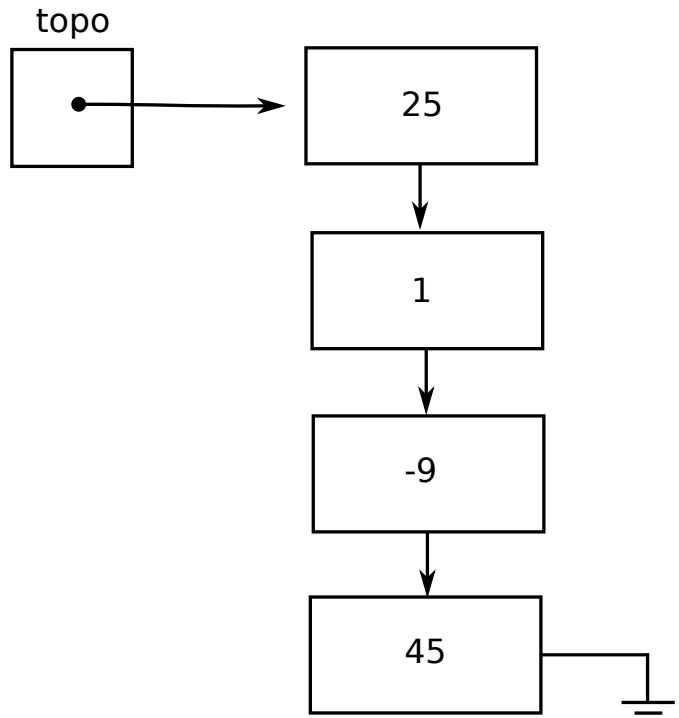
```
void PilhaCont::empilha(int val)
{
  if(topo == (max - 1)){
    cout << "ERRO: Vetor Cheio!" << endl;
    exit(2);
  }
  topo = topo + 1;
  vet[topo] = val;
}

int PilhaCont::desempilha()
{
  if(topo == -1){
    cout << "ERRO: Pilha vazia!" << endl;
    exit(1);
  }
  topo = topo - 1;
  return vet[topo + 1];
```

Pilhas

Representação encadeada

- Uma pilha de valores inteiros com n nós pode ser **representada** através de **uma lista simplesmente encadeada**.



- Portanto, será necessário definir dois TADs: `No` e `PilhaEncad`.

Pilhas

Representação encadeada

- O TAD `No` possui a mesma implementação usada anteriormente para uma lista simplesmente encadeada.

```
class No
{
 public:
   // ...

 private:
   int info; // informacao do No
   No *prox;
};
```

Pilhas

Representação encadeada

```
class PilhaEncad
{
 public:
   PilhaEncad();
   ~PilhaEncad();

   int getTopo();
   void empilha(int val);
   int desempilha();
   bool vazia();

 private:
   No * topo;
};
```

Pilhas

Representação encadeada

```
PilhaEncad::PilhaEncad()
{
  topo = NULL;
}

PilhaEncad::~PilhaEncad()
{
  No *p = topo;
  while(topo != NULL)
  {
    topo = p->getProx();
    delete p;
    p = topo;
  }
}
```

Pilhas

Representação encadeada

```
int PilhaEncad::getTopo()
{
  if(topo != NULL)
    return topo->getInfo();
  else
  {
    cout << "ERRO: Pilha vazia!" << endl;
    exit(1);
  }
}

void PilhaEncad::empilha(int val)
{
  No *p = new No();
  p->setInfo(val);
  p->setProx(topo);
  topo = p;
}
```

Pilhas

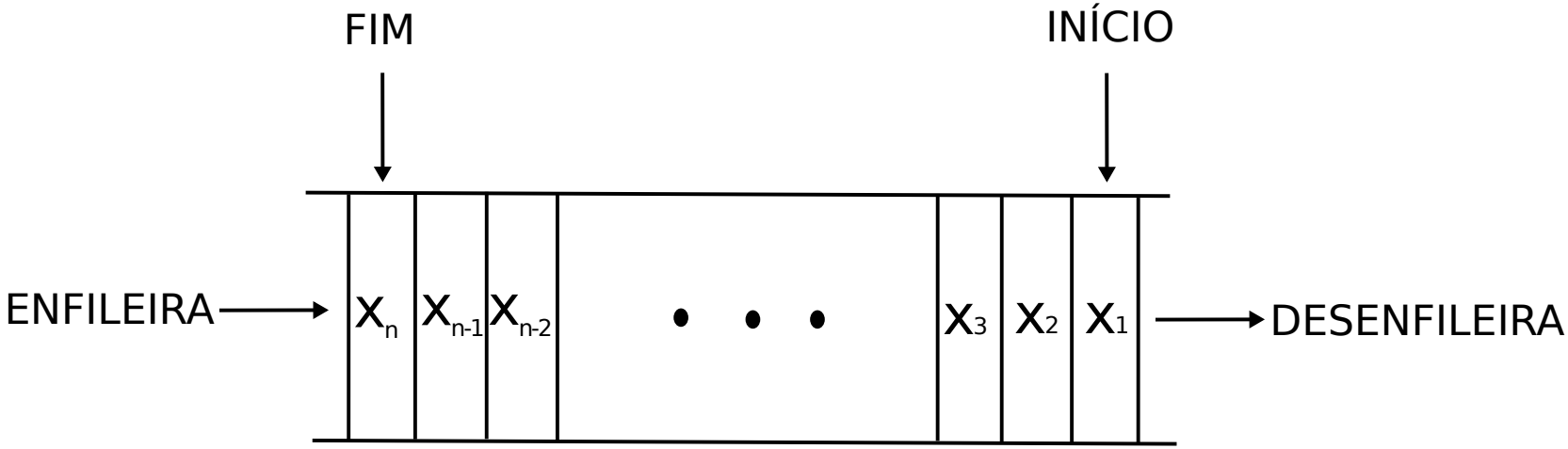
```
int PilhaEncad::desempilha()
{
  No *p;
  int val;
  if(topo != NULL)
  {
    p = topo;
    topo = p->getProx();
    val = p->getInfo();
    delete p;
  }
  else
    exit(1);
  return val;
}

bool PilhaEncad::vazia()
{
  return (topo == NULL)
```

Filas

- Assim como as pilhas, as filas também possuem uma disciplina de acesso bem definida:
  - A fila é uma lista onde todas as **inclusões são feitas em uma extremidade** e todas as **exclusões na outra extremidade**.
  - Usa-se o critério de retirar em primeiro lugar o primeiro nó incluído na lista.
  - Este critério é chamado de **FIFO** (**F**irst **I**n **F**irst **O**ut).
  - Esta estrutura é equivalente a uma fila de banco.

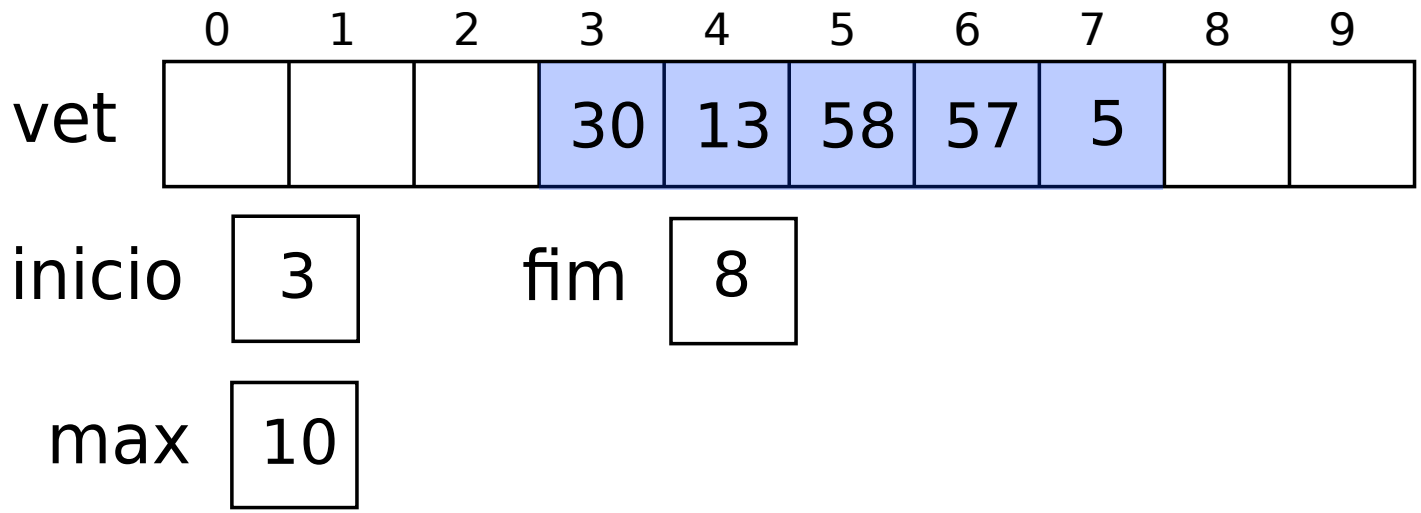
Filas



- Representações:
  - Contígua
  - Encadeada

Filas

- Representação por contiguidade dos nós.
- Uma fila contígua onde já ocorreram $(n+1)$ entradas e k saídas de nós, teria a seguinte representação esquemática:



- Sendo *inicio* o índice do vetor que contém o nó do começo e *fim* o índice do nó que está no final.

TAD Fila Contígua

```
class FilaCont
{
 private:
  int max;    // maximo de elementos do vetor
  int inicio;// indice do No que esta no inicio
  int fim;    // indice da posicao apos o ultimo No
  int *vet;   // vetor que armazena a fila

 public:
  FilaCont(int tam);
  ~FilaCont();
  int getInicio();          // valor do No no inicio
  void enfileira(int val);// insere No no fim
  int desenfileira();       // elimina No do inicio
  bool vazia();             // verifica se esta vazia
};
```

TAD Fila Contígua

- Construtor e destrutor

```
FilaCont::FilaCont(int tam)
{
    max = tam;
    inicio = 0;
    fim = 0;
    vet = new int[max];
}

FilaCont::~FilaCont()
{
    delete [] vet;
}
```

TAD Fila Contígua

```
int FilaCont::getInicio()
{
    if (!vazia())
        return vet[inicio];
    else {
        cout << "ERRO: Fila vazia" << endl;
        exit(1);
    }
}

void FilaCont::enfileira(int val)
{
    if(fim == max) // fila cheia
    {
        cout << "ERRO: Fila cheia" << endl;
        exit(2);
    }
    vet[fim] = val;
    fim = fim + 1;
```

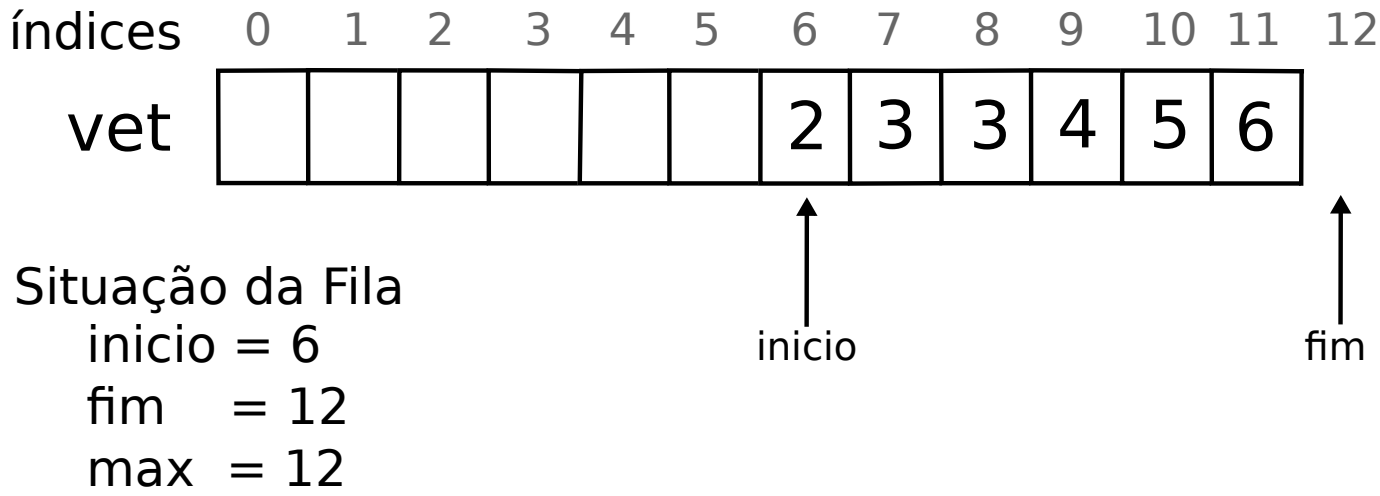
TAD Fila Contígua

```
int FilaCont::desenfileira()
{
    if(!vazia())
    {
        inicio = inicio + 1;
        return vet[inicio - 1];
    }
    cout << "ERRO: Fila vazia" << endl;
    exit(1);
}

bool FilaCont::vazia()
{
    return (inicio==fim);
}
```

Problemas do TAD Fila Contígua

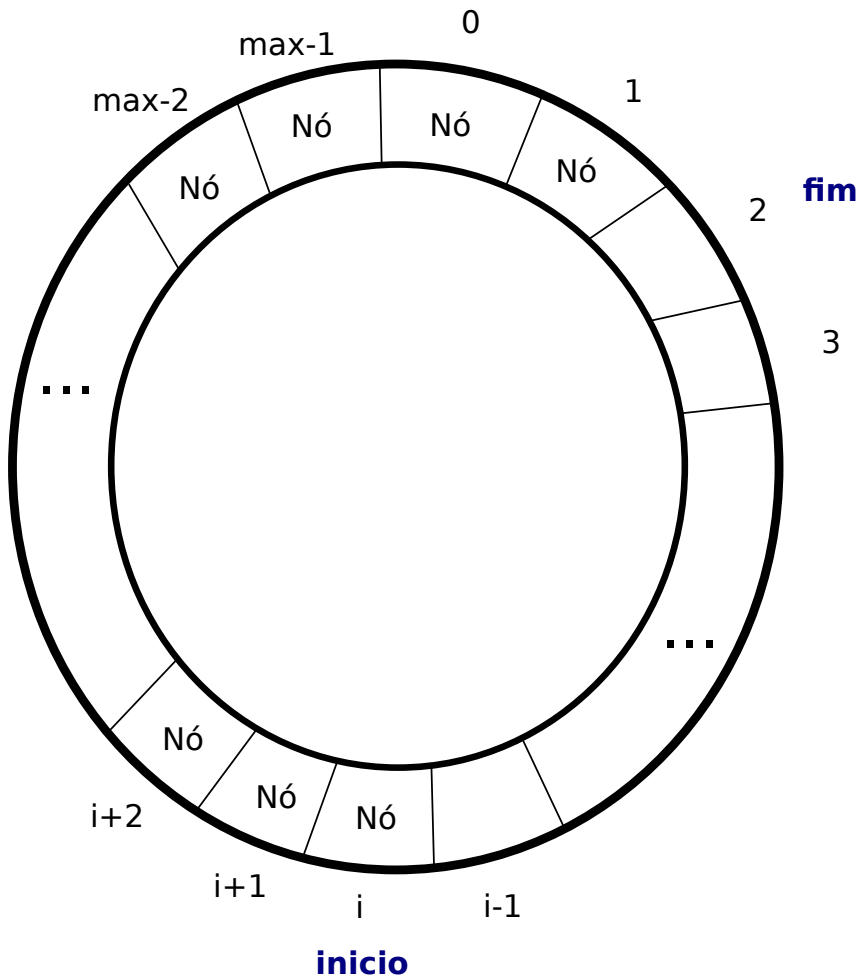
- O TAD `FilaCont` apresenta algumas restrições.



- Quando *fim* = *max* não se pode mais inserir na fila.
- Porém, se *inicio* > 0 então existe espaço no início da fila.
- Esse espaço deveria ser usado para inserir mais nós na fila.
- A solução mais comum é utilizar uma representação circular do vetor que representa a fila.

TAD Fila Circular

- Ideia: todos os incrementos em *inicio* e *fim*, quando sai e entra nó na fila, devem ser realizados módulo *max*.
- TAD Fila Circular:



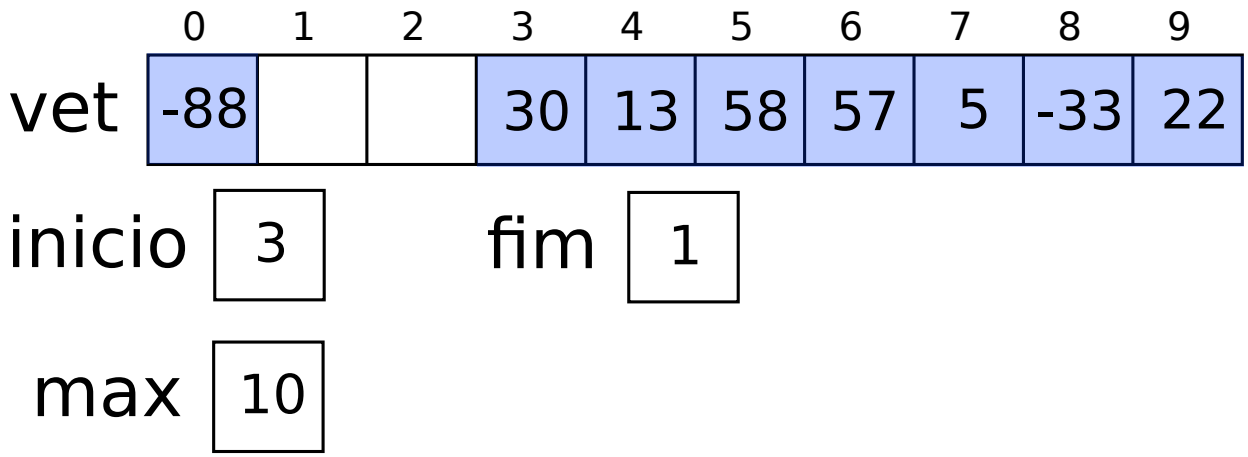
Situação da Fila

inicio = i

fim = 2

TAD Fila Circular

- Exemplo:



- E se mais dois valores fossem inseridos na fila?
- Nesse caso, `inicio=fim` e assim a condição de fila vazia se confundiria com a de fila cheia.
- Para evitar esse problema usa-se um contador `n` para saber quantos valores estão de fato na fila.

TAD Fila Circular

```
class FilaCirc
{
 private:
  int max;     // maximo de elementos do vetor
  int inicio; // indice do No que esta no inicio
  int fim;     // indice do No que esta no fim
  int n;       // numero atual de elementos do vetor
  int *vet;    // vetor que armazena a fila

  int inc(int ind); // incrementa indice

 public:
  FilaCirc(int tam);
  ~FilaCirc();
  int getInicio();      // consulta No do inicio
  void enfileira(int val); // insere No no fim
  int desenfileira();     // elimina No do inicio
  bool vazia();           // verifica se esta vazia
```

TAD Fila Circular

Incremento circular

- A operação de *incremento circular* para o TAD Fila Circular pode ser implementada da seguinte forma:

```
int FilaCirc::inc(int ind)
{
  if (ind == max-1)
    return 0;
  else
    return ind+1;
}

// ou

int FilaCirc::inc(int ind)
{
  return (ind+1) % max;
}
```

TAD Fila Circular

Incremento circular

- Construtor: precisa ser modificado para incluir n, o número de nós presentes na fila.

```
FilaCirc::FilaCirc(int tam)
{
    max = tam;              // capacidade maxima
    n = 0;                  // numero de nos na fila
    inicio = fim = 0;       // inicio, fim
    vet = new int[max]; // vetor com os dados
}

FilaCirc::~FilaCirc()
{
    delete [] vet;
}
```

TAD Fila Circular

- Operação `getInicio()`: sem alterações com relação ao TAD anterior `FilaCont`.
- A condição de fila vazia agora é diferente.
- É preciso verificar o valor de n, que representa o número de nós inseridos na fila:
  - se $n = 0$: a fila está **vazia**;
  - se $n = max$: a fila está **cheia**, não é possível inserir mais nós.

```
bool FilaCirc::vazia()
{
  return (n==0);
}
```

- Para o TAD `FilaCirc` só é preciso mudar as operações de entrada e saída da fila usando o **incremento circular** .

TAD Fila Circular

```
void FilaCirc::enfileira(int val)
{
    if(n == max) {
        cout << "ERRO: Fila cheia" << endl;
        exit(1);
    }
    vet[fim] = val;
    fim = inc(fim);
    n = n + 1;
}
```

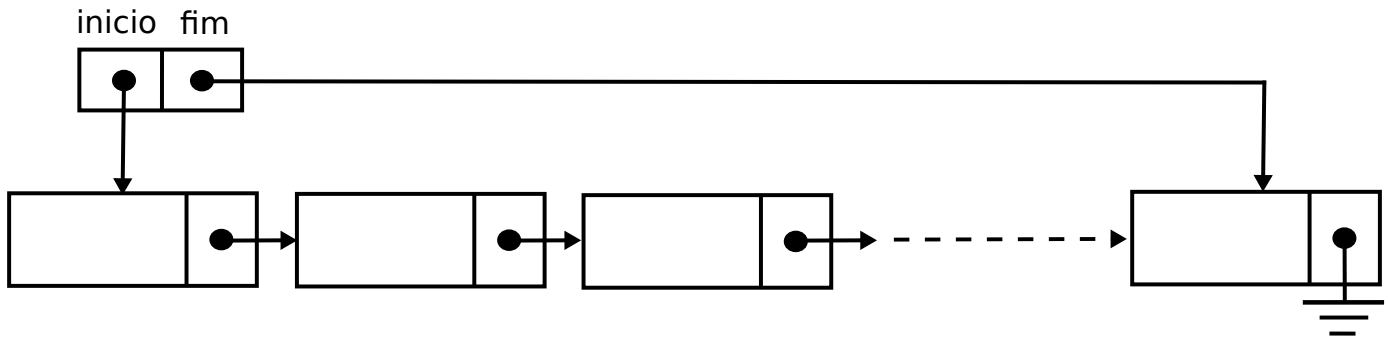
TAD Fila Circular

```
int FilaCirc::desenfileira()
{
    if(!vazia())
    {
        int val = vet[inicio];
        inicio = inc(inicio);
        n = n - 1;
        return val;
    }
    cout << "ERRO: Fila vazia" << endl;
    exit(1);
}
```

Fila

Encadeamento dos nós

- Representação por encadeamento dos nós.
- Esquematicamente, seja uma fila encadeada com n nós:



- Para representar uma fila dessa forma, é preciso definir dois TADs:
  - `No`
  - `FilaEncad`

TAD Fila Encadeada

- Definição da classe do TAD `No` (No.h)
- Pode-se utilizar o mesmo nó definido anteriormente para uma lista simplesmente encadeada ou pilha encadeada.

```
class No
{
 public:
   // ...

 private:
   int info; // informacao do No
   No *prox;
};
```

TAD Fila Encadeada

- Definição da classe do TAD `FilaEncad`
- Arquivo: FilaEncad.h

```
class FilaEncad
{
 private:
  No *inicio; // ponteiro para No do comeco
  No *fim;    // ponteiro para No do fim

 public:
  FilaEncad();
  ~FilaEncad();
  int getInicio();     // retorna No do inicio
  void enfileira(int val); // insere No no fim
  int desenfileira(); // elimina No do inicio
  bool vazia();          // verifica se esta vazia
};
```

TAD Fila Encadeada

- Arquivo: FilaEncad.cpp

```
FilaEncad::FilaEncad()
{
  inicio = NULL;
  fim = NULL;
}

FilaEncad ::~FilaEncad ()
{
  No *p = inicio;
  while(inicio != NULL)
  {
    inicio = p->getProx();
    delete p;
    p = inicio;
  }
}
```

TAD Fila Encadeada

```
bool FilaEncad::vazia()
{
  // verifica se fila esta vazia
  if(inicio == NULL)
    return true;
  else
    return false;
}

int FilaEncad::getInicio()
{
  if(inicio != NULL)
    return inicio->getInfo();
  else
  {
    cout << "ERRO: Fila vazia!" << endl;
    exit(1);
  }
```

TAD Fila Encadeada

```
void FilaEncad::enfileira(int val)
{
  No *p = new No();
  p->setInfo(val);
  p->setProx(NULL);

  if(fim == NULL)
    inicio = p;          // insere o primeiro No
  else
    fim->setProx(p);     // liga No p na fila

  fim = p;               // No p passa a ser o ultimo
}
```

TAD Fila Encadeada

```
int FilaEncad::desenfileira()
{
  No *p;
  if(inicio != NULL)
  {
    p = inicio;
    inicio = p->getProx();

    if(inicio == NULL)
      fim = NULL;          // a fila esvaziou

    int val = p->getInfo();
    delete p;              // exclui o No do inicio
    return val;
  }
  cout << "ERRO: Fila vazia" << endl;
  exit(1);
}
```

Exercícios

1. Desenvolver uma função para inverter o sentido de uma **fila** representada por uma lista encadeada.

2. Considerando uma **pilha** representada por uma lista encadeada onde cada nó possui um valor inteiro, desenvolver um procedimento para desempilhar os seus nós montando uma lista circular (duplamente encadeada).

3. O gerente de desenvolvimento de aplicações de uma empresa observou que diversos programas trabalham com **filas** encadeadas que devem ser gravadas em memória auxiliar. Para isso é necessário representar a fila contiguamente. Desenvolver um procedimento para fazer esta transformação sabendo que cada nó possui um valor inteiro e que uma fila possui no máximo 500 nós.

Exercícios

4. Desenvolver um procedimento para fazer a transformação inversa da anterior, i.e., de fila contígua para fila encadeada.
5. Desenvolver um procedimento para montar duas **filas** contíguas, sendo que uma deve conter somente números inteiros negativos, e a outra os demais números. Os números inteiros serão fornecidos através de uma **pilha** encadeada em que cada nó possui um número inteiro e qualquer. Considerar que a pilha possui no máx. 100 nós.
6. Desenvolver um programa para ler diversos valores inteiros e montar uma **fila** encadeada, usando o TAD FilaEncad e atendendo as seguintes características:
   a) O último valor a ser lido é um FLAG com $X = 0$.
   b) Na fila só podem entrar nós com $X > 0$ e maior que a informação X do último nó já incluído na fila.
   c) Para cada valor $X < 0$ lido, retirar um nó da fila, se não for vazia.
   d) No final, imprimir todos os nós da fila