Trabalho de Estrutura de Dados

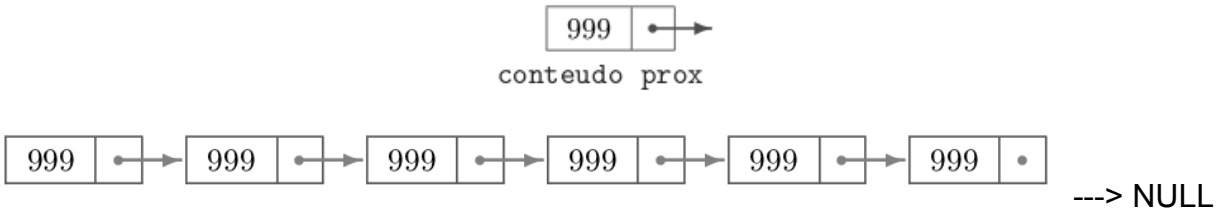
Ana Cássia, Daniella, Gabriel Dias, Guilherme Cassimiro, Heron, Mateus Delai
(Equipe 4).

Biopark Connect 2022

LISTA ENCADEADA SIMPLES

Uma lista encadeada é uma representação de uma sequência de objetos, todos do mesmo tipo, na memória RAM do computador. Cada elemento da sequência é armazenado em uma célula da lista: o primeiro elemento na primeira célula, o segundo na segunda, e assim por diante. Na lista encadeada simples, cada célula contém um objeto e o endereço da célula seguinte.
Estrutura:

```
struct reg {
    int       conteudo;
    struct reg *prox;
  };
```



O endereço de uma lista encadeada é o endereço de sua primeira célula.

Busca em uma lista encadeada:

A função de busca exemplificada a seguir recebe um inteiro x e uma lista encadeada le de inteiros e devolve o endereço de uma célula que contém x. Se tal célula não existe, devolve NULL.

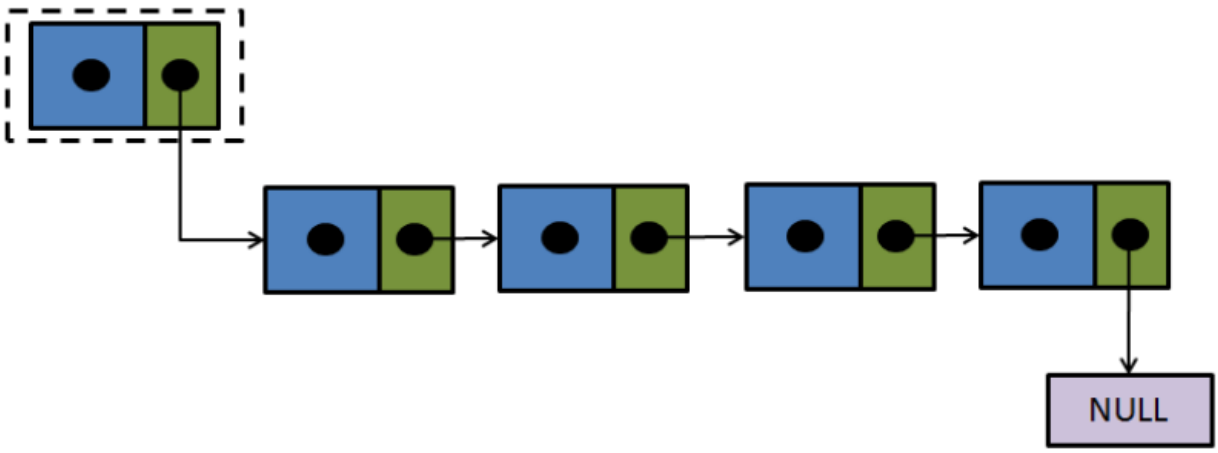
```
celula *
busca (int x, celula *le)
{
  celula *p;
  p = le;
  while (p != NULL && p->conteudo != x)
    p = p->prox;
  return p;
}
```

Inserção em Lista Encadeada:

Considere o problema de inserir uma nova célula em uma lista encadeada. Suponha que quero inserir a nova célula entre a posição apontada por p e a posição seguinte. (É claro que isso só faz sentido se p é diferente de NULL.)

```
// Esta função insere uma nova celula
// em uma lista encadeada. A nova celula
// tem conteudo x e é inserida entre a
// celula p e a celula seguinte.
// (Supõe-se que p != NULL.)

void
insere (int x, celula *p)
{
  celula *nova;
  nova = malloc (sizeof (celula));
  nova->conteudo = x;
  nova->prox = p->prox;
  p->prox = nova;
}
```

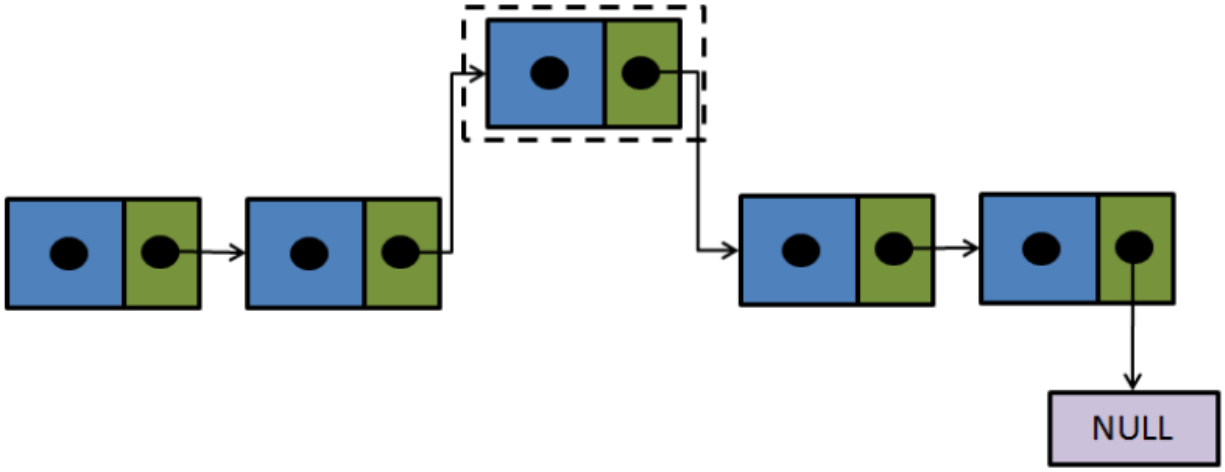
Inserção no início da lista:



1. Criar a nova célula a ser inserida;
2. Apontar o ponteiro **proximo** do no a ser inserido para o primeiro nó da lista original (novo_no.proximo = inicio);
3. Apontar o inicio para o novo nó inserido

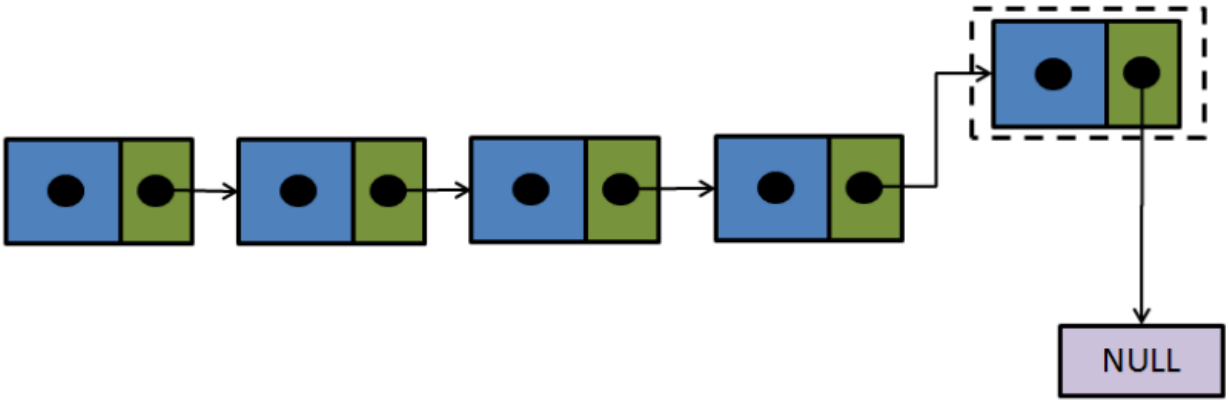
Inserção no meio da lista:

A inserção de um elemento no meio da lista ou em uma posição arbitrária independe do fato da lista possuir ou não cabeçalho. A idéia geral é encontrar a posição na qual a célula corrente deve ser inserida e então identificar as células anterior e posterior. Uma vez identificadas tais células basta que os ponteiros sejam ajustados.



Inserção no fim da lista:

Tal inserção ocorre de maneira muito similar ao caso anterior. A única diferença é que no proximo deve apontar para NULL.



Remoção em Lista Encadeada:

Os elementos de uma lista podem ser removidos, a semelhança das regras de inserção, de três maneiras distintas:

- Remoção do primeiro elemento da lista;
- Remoção do último elemento da lista;
- Remoção de um elemento em uma posição arbitrária.

As funções de remoção são muito similares as de inserção.

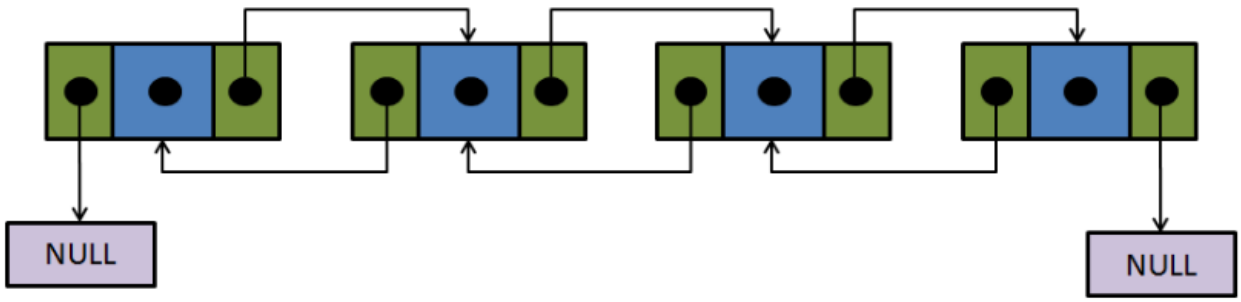
```
// Esta função recebe o endereço p de uma
// celula de uma lista encadeada e remove
// da lista a celula p->prox. A função supõe
// que p != NULL e p->prox != NULL.

void
remove (celula *p)
{
  celula *lixo;
  lixo = p->prox;
  p->prox = lixo->prox;
  free (lixo);
}
```

LISTA DUPLAMENTE ENCADEADA

Uma lista duplamente encadeada é assim denominada pela forma como os nós que compõem a lista são ligados. Ao contrário das listas simples, existem dois caminhos possíveis para se percorrer a lista. Pode-se pesquisar a lista duplamente encadeada em sentido normal e em sentido inverso. Isso se da pelo fato de que os elementos da lista possuem dois ponteiros, um que aponta para o próxim o nó e outro que aponta para o anterior. Adicionalmente, o primeiro nó terá seu ponteiro **anterior** apontando para NULL e o último nó terá seu ponteiro **proximo** apontando igualmente para NULL.

Estrutura de Dados de uma Lista Duplamente Encadeada

```
class No {            // Nó da lista duplamente encadeada
   int informacao;    // Informação que se quer armazenar
   No proximo;        // Ponteiro para o próximo nó
   No anterior;       // Ponteiro para o nó anterior
}
class ListaDupla {    // Lista duplamente encadeada
   No inicio;         // Ponteiro para o primeiro nó da lista
   No fim;            // Ponteiro para o último nó da lista
   int tamanho;       // Quantidade de elemntos na lista
}
```



LISTA ENCADEADA CIRCULAR

Lista Circular é um tipo de lista simples ou duplamente encadeada que possui um numero infinito de elementos .

Sua estrutura consiste em um campo para armazenar as informaçoes , um ponteiro direcionado para o proximo elemento da lista e um ponteiro auxiliar que define o comeco da lista.

Vantagens :

Melhor utlização dos recursos de memoria.

Inserção e remoção de elementos ocorre sem deslocar os elementos.

Utilizada quando a inserçao/remoção de elementos é mais frequente

Desvantagem :

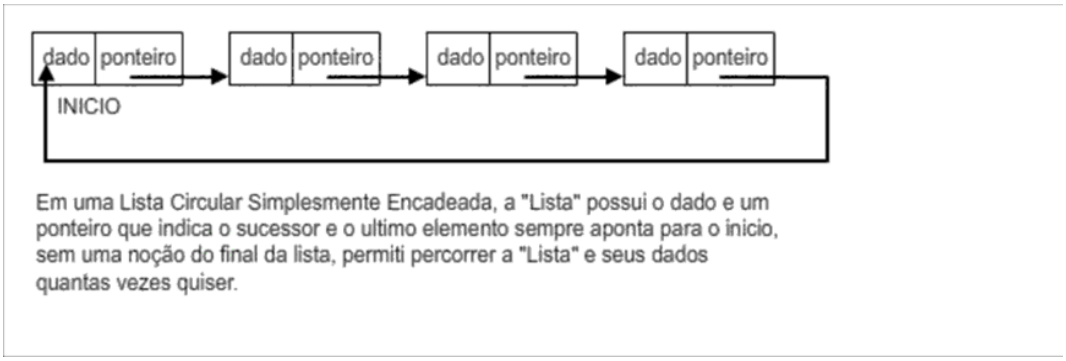
Acesso aos elementos ocorre de forma indireta.

Para acessar um elemento é preciso percorrer toda a lista. Busca se torna mais custosa em termos de uso de memoria

Insercao:

Caso a lista esteja vazia a inserção acontece difinindo esse elemento como inicio da lista e seu nó aponta para ele mesmo.

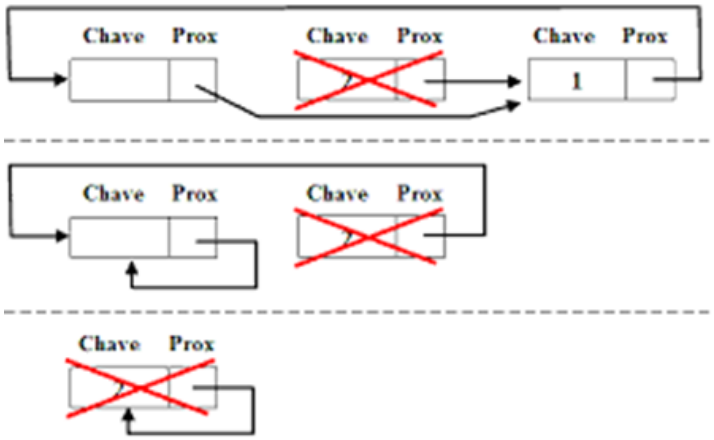
Para inserção em lista que ja existe elementos , o novo elemento apontara para o primeiro elemento da lista e o ultimo elemento aponta-ra para o novo elemento e depois o inicio da lista devera apontar para o novo elemento.



Em uma Lista Circular Simplesmente Encadeada, a "Lista" possui o dado e um ponteiro que indica o sucessor e o ultimo elemento sempre aponta para o inicio, sem uma noção do final da lista, permiti percorrer a "Lista" e seus dados quantas vezes quiser.

Remoção:

Basicamento a remoção de elemento acontece mudando a direção do ponteiro anterior ao elemento a ser removido apos isso é feito um free no elemento.



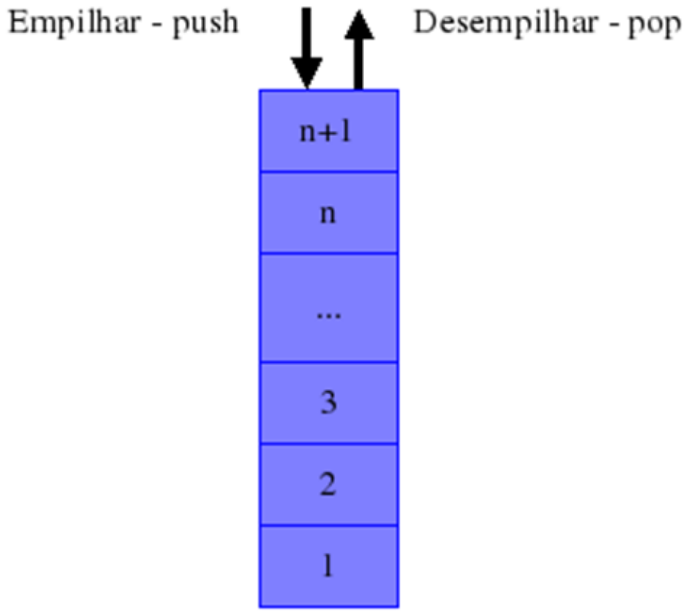
Busca:

A consulta pode ocorrer por posição ou elemento para isso sera necessario varrer todos os elementos da lista procurando o elemento desejado.

PILHA

Pilhas são estruturas de dados em que só é possível inserir um novo elemento no final da pilha e só é possível remover um elemento do final da pilha.

As pilhas seguem um protocolo onde o último a entrar é o primeiro a sair e geralmente são implementadas com arranjos (Arranjos são sequências de **elementos de um mesmo tipo**, armazenados em **posições contíguas de memória**). Essa política é conhecida pela sigla LIFO (= *Last-In-First-Out*).



Fonte: https://www.cos.ufrj.br/~rfarias/cos121/pilhas.html

São exemplos de uso de pilha em um sistema:

- Funções recursivas em compiladores;
- Mecanismo de desfazer/refazer dos editores de texto:

Quando as teclas ctrl + z ou o botão desfazer são pressionador, a última ação empilhada é desempilhada, desfeita e empilhada na pilha da opção refazer.

- Navegação entre páginas Web;

Todas as operações em uma pilha podem ser imaginadas como as que ocorre numa pilha de pratos em um restaurante ou como num jogo com as cartas de um baralho:

- criação da pilha (informar a capacidade no caso de implementação sequencial - vetor);
- empilhar (push) - o elemento é o parâmetro nesta operação;
- desempilhar (pop);
- mostrar o topo;
- verificar se a pilha está vazia (isEmpty);
- verificar se a pilha está cheia (isFull - implementação sequencial - vetor).

Estruturas:

A criação de um algoritmo de pilha se dá através de duas estruturas, uma estrutura nó com dois elementos, um valor inteiro e um ponteiro para outro nó; e a estrutura pilha, que possui um elemento do tipo nó chamado topo, e um inteiro para o tamanho.

Código em C:

```
typedef struct no{
        int valor;
   struct no *proximo;
}No;


typedef struct{
        No *topo;
        int tam;
}Pilha;
```

Operação Empilhar:

Para a operação empilhar ou inserir, o procedimento receberá um ponteiro para a pilha e o elemento a ser inserido. Um novo nó é alocado dinamicamente e inserido no topo da lista, atualizando o ponteiro para o topo em seguida.

Código em C:

```
// operação push

void empilhar(Pilha *p, int x){

        No *no = malloc(sizeof(No));

   no->valor = x;

   no->proximo = p->topo;

   p->topo = no;

}
```

Operação Desempilhar:

A função desempilhar retorna o ponteiro para o topo desempilhado, ou nulo caso a pilha esteja vazia. Antes de retornar o nó desempilhado, o topo é atualizado par ao nó seguinte.

```
/*  operação pop

        retorna o topo da pilha ou NULL

*/

No* desempilhar(Pilha *p){

        No *no = NULL;

        if(p->topo){

        no = p->topo;

        p->topo = no->proximo;

        }

        return no;

}
```

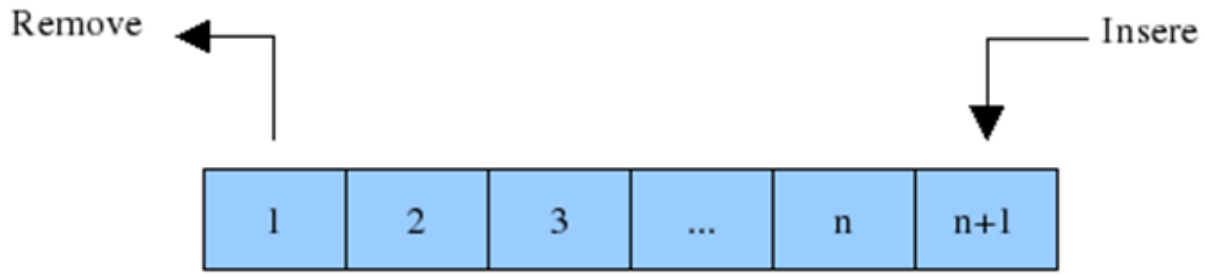
FILA

Uma fila é uma estrutura de dados dinâmica que admite remoção de elementos e inserção de novos objetos.

Filas são listas em que todas as inserções ocorrem em uma das extremidades e todas as remoções ocorrem na outra, essa política é conhecida pela sigla FIFO (= First-in-First-out).

Início da fila: extremidade onde ocorrem as remoções.

Final da fila: extremidade onde ocorrem as inserções.



São exemplos de uso de fila em um sistema:

- Controle de documentos para impressão;
- Troca de mensagem entre computadores numa rede;
- etc.

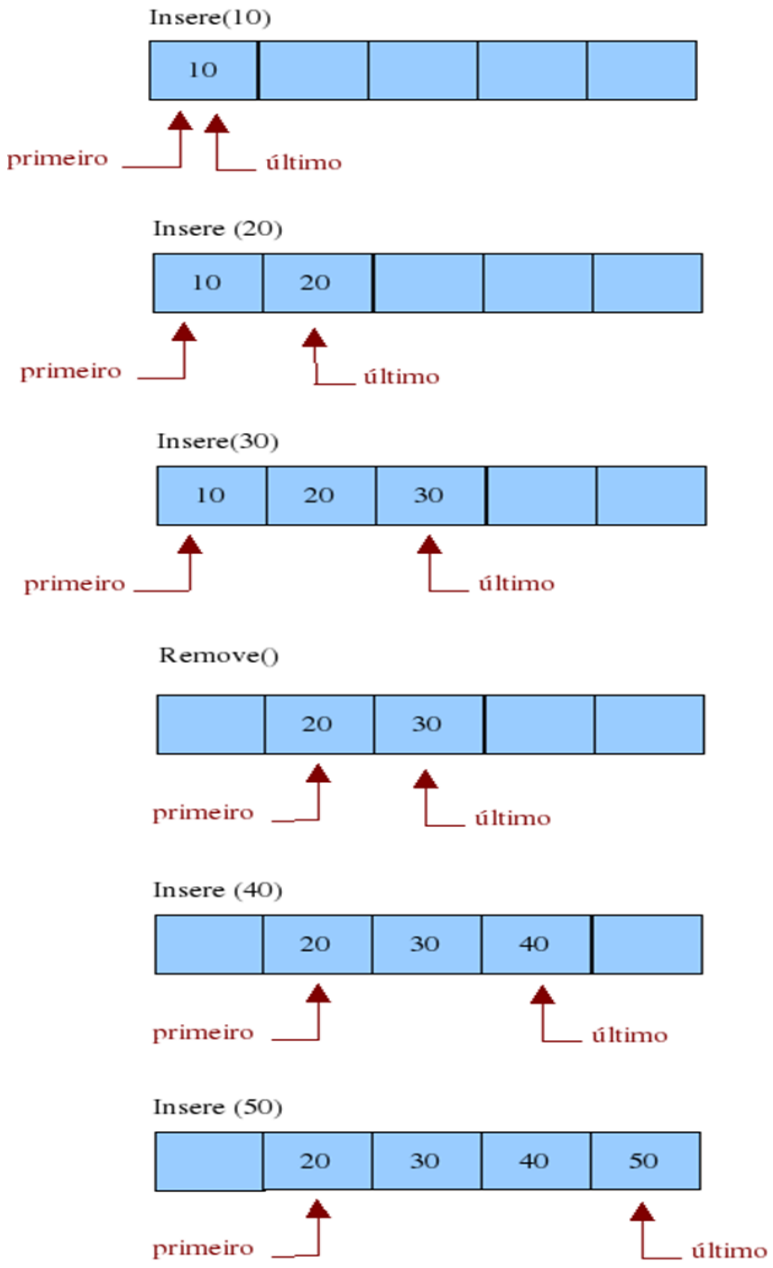
A implementação de filas pode ser realizada através de vetor (alocação do espaço de memória para os elementos é contígua) ou através de listas encadeadas (próxima aula).

Operações com Fila:

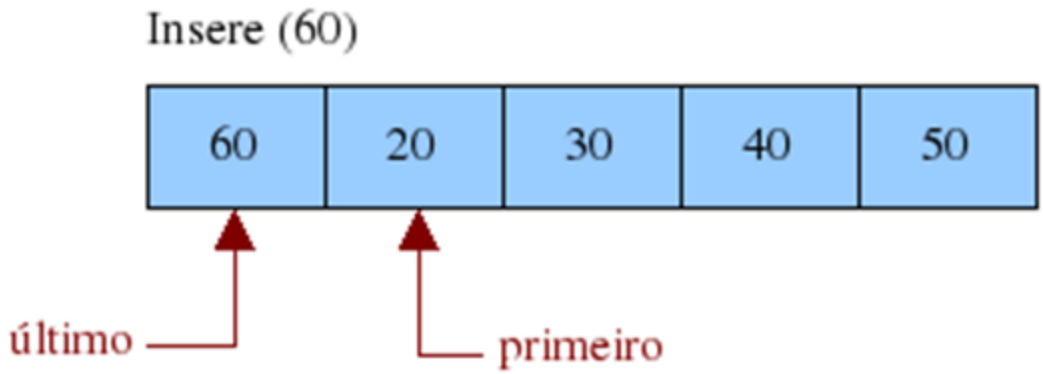
Todas as operações em uma fila podem ser imaginadas como as que ocorre numa fila de pessoas num banco, exceto que os elementos não se movem na fila, conforme o primeiro elemento é retirado. Isto seria muito custoso para o computador. O que se faz na realidade é indicar quem é o primeiro.

- criação da fila (informar a capacidade no caso de implementação sequencial - vetor);
- enfileirar (enqueue) - o elemento é o parâmetro nesta operação;
- desenfileirar (dequeue);
- mostrar a fila (todos os elementos);
- verificar se a fila está vazia (isEmpty);
- verificar se a fila está cheia (isFull - implementação sequencial - vetor).

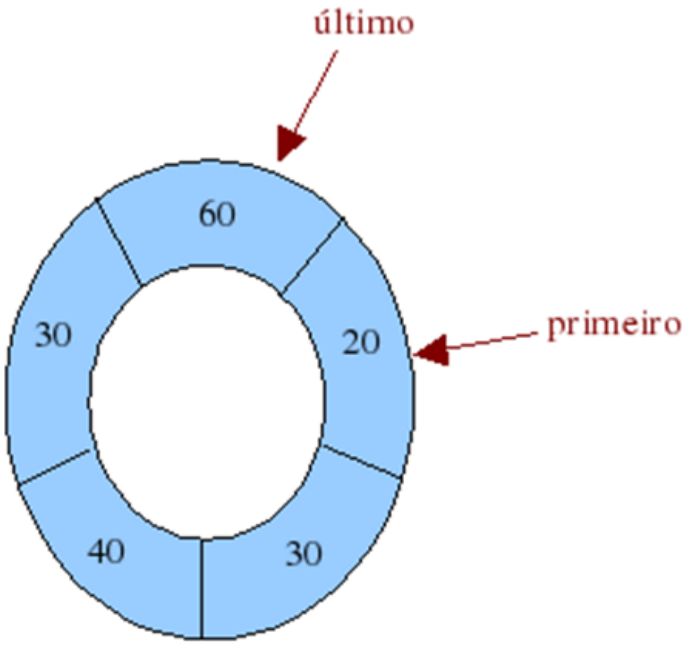
Supondo uma fila com capacidade para 5 elementos:



Para evitar problemas de não ser capaz de inserir mais elementos na fila, mesmo quando ela não está cheia, as referências primeiro e último circundam até o início do vetor, resultando numa fila circular.



Desta forma a fila simula uma representação circular:



Para a implementação circular podemos definir uma função auxiliar que é responsável por incrementar o valor de um índice. Essa função recebe o valor do índice atual e retorna o valor do índice incrementado com o incremento circular.

```
int incr (int i) {
   if (i == N-1)
        return 0;
   else
        return i+1;
}
```

Uma fila pode ser declarada utilizando a seguinte função:

```
#define N 100 // tamanho da fila
struct fila {
   int ini, fim;  // inicio e fim da fila
   float vet[N]; // vetor com os elementos da fila
};
```

typedef struct fila Fila;

A função para criar a fila aloca dinamicamente essa estrutura e inicializa a fila como sendo vazia, isto é, com os índices ini e fim iguais entre si (no caso, usamos o valor zero).

```
Fila* cria (void) {
   Fila* f = (Fila*) malloc(sizeof(Fila));  // Cria ponteiro para a fila
   f->ini = f->fim = 0;                     // inicializa fila vazia
   return f;
}
```

Para inserir um elemento na fila, usamos a próxima posição livre do vetor, indicada por fim. Devemos ainda assegurar que há espaço para a inserção do novo elemento, tendo em vista que trata-se de um vetor com capacidade limitada.

```
void insere (Fila* f, float v) {
   if (incr(f->fim) == f->ini) {
        /* fila cheia: capacidade esgotada  */
        printf("A Fila está cheia. Nao e possivel inserir mais elementos.\n");
        exit(1);    /* finaliza funcao retornando 1 */
   }

   /* insere elemento na próxima posição livre */
   f->vet[f->fim] = v;          // Insere o elemento no fim da fila usando o vetor vet indexado por f->fim
   f->fim = incr(f->fim);       // Incrementa a posicao do fim da fila em 1
}
```

A função para retirar o elemento do início da fila fornece o valor do elemento retirado como retorno. Podemos também verificar se a fila está ou não vazia.

```
float retira (Fila* f) {
   float v;
   if (vazia(f)) {
        printf("A Fila esta vazia.\n");
        exit(1);     // finaliza funcao retornando 1
   }
```

```
   /* retira elemento do início */
   v = f->vet[f->ini];  // Obtem o elemento do inicio da fila
   f->ini = incr(f->ini); // Incrementa o indice para o inicio da fila
   return v;              // retorna o valor do elemento que estava no inicio da fila
}
```

Finalmente, a função para liberar a memória alocada pela fila pode ser:

```
void libera (Fila* f) {
   free(f);
}
```

Referências

http://www.ic.uff.br/~fabio/Aula-pilhas.pdf

https://algoritmosempython.com.br/cursos/algoritmos-python/estruturas-dados/pilhas/

https://www.cos.ufrj.br/~rfarias/cos121/pilhas.html

https://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html

https://www.treinaweb.com.br/blog/o-que-e-e-como-funciona-a-estrutura-de-dados-pilha

https://wagnergaspar.com/estrutura-de-dados-do-tipo-pilha/

https://www.cos.ufrj.br/~rfarias/cos121/filas.html

https://www.ime.usp.br/~pf/algoritmos/aulas/fila.html

https://sites.google.com/site/proffdesiqueiraed/aulas/aula-7---filas

http://www.ic.uff.br/~fabio/Aula-filas.pdf

https://www.ime.usp.br/~pf/algoritmos/aulas/lista.html

https://saulo.arisa.com.br/wiki/index.php/Listas_Encadeadas_Simples