CRISTYAN MORAIS, GABRIEL NAVA, JOSÉ KLAK, LUCAS PASSAROTE,
MARCUS SILVÉRIO, PAULA VALENTE

ESTRUTURA DE DADOS:

Lista, fila e pilha

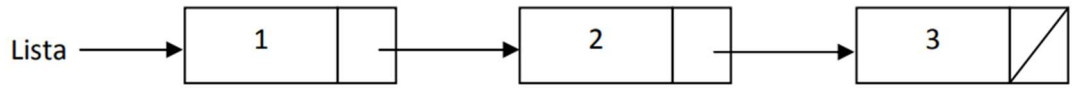
Toledo – PR

2022

Listas simplesmente encadeadas

O que são?

Uma lista simplesmente encadeada é uma sequência de nós, onde cada nó contém uma informação de algum tipo de dado e o endereço do nó seguinte. Vamos supor que a informação seja uma variável do tipo **int**.



A lista é um ponteiro para o primeiro nó da lista que contém o valor "1" e um ponteiro para o próximo nó. Já o último nó da lista aponta para NULL, indicando fim da lista.

Como são adicionados novos elementos (nós) na lista?

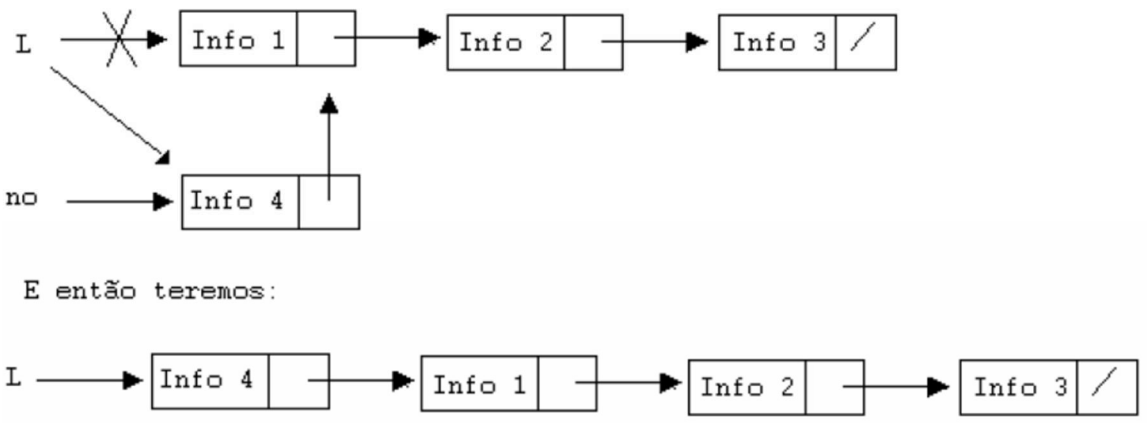
Após a criação do novo elemento (nó), existem três casos para sua inserção na lista:

O nó é inserido **no início** da lista;

O nó é inserido **no meio** da lista;

O nó é inserido **no fim** da lista.

Quando o nó é inserido **no início da lista** deve-se fazer o campo "próximo" deste nó apontar para o primeiro elemento da lista, e em seguida fazer a lista apontar para este nó recém-criado.

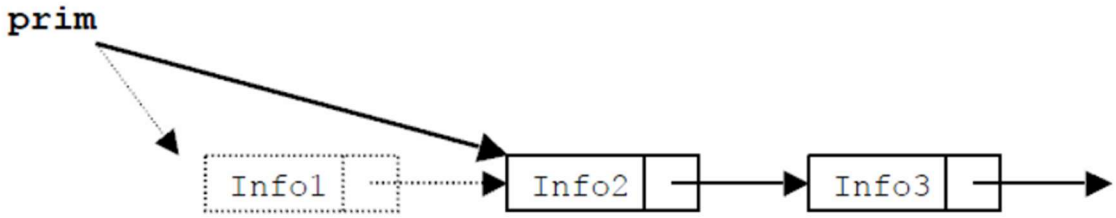


Quando o nó é inserido no **fim da lista**, deve-se fazer o campo "próximo" do último nó da lista apontar para o elemento que se deseja inserir. Para tanto, devemos percorrer a lista até encontrar o último elemento. Lembrando que o último elemento sempre aponta para NULL.

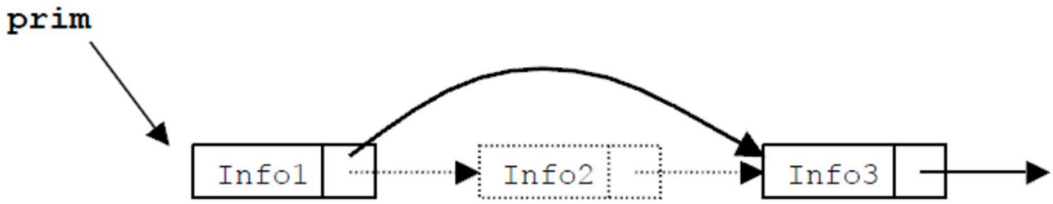
Quando o nó é inserido **no meio da lista**, deve-se informar a partir de qual nó o novo elemento será inserido. Ou seja, vamos utilizar o campo "info" para fazer uma pesquisa na lista, e após encontrá-lo, inserir o novo elemento.

Como são removidos elementos (nós) da lista?

Caso o elemento a ser retirado **seja o primeiro da lista**, devemos fazer com que o novo valor da lista passe a ser o ponteiro para o segundo elemento, e então podemos liberar o espaço alocado para o elemento que queremos retirar, dessa forma:



Caso o elemento a ser removido **estiver no meio da lista**, devemos fazer com que o elemento anterior a ele passe a apontar para o elemento seguinte, e então podemos liberar o elemento que queremos retirar, dessa forma:

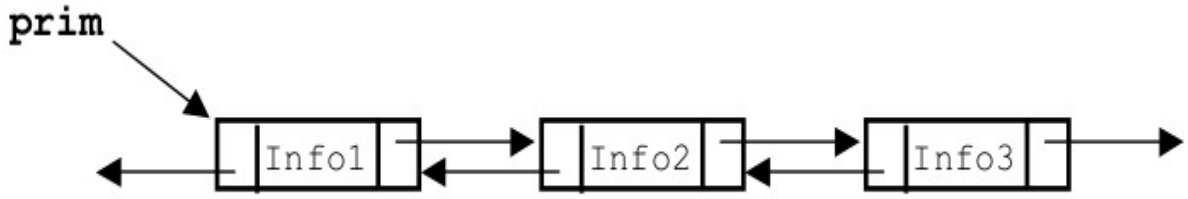


Como são feitas as buscas por um elemento (nó)?

É necessário percorrer a lista a partir do primeiro elemento até onde for desejado, ou seja, até achar o nó (elemento) procurado ou chegar ao final da lista.

Listas duplamente encadeadas

Nelas, cada elemento tem um ponteiro para o próximo elemento e um ponteiro para o elemento anterior. Desta forma, dado um elemento, podemos acessar ambos os elementos adjacentes: o próximo e o anterior. Se tivermos um ponteiro para o último elemento da lista, podemos percorrer a lista em ordem inversa, bastando acessar continuamente o elemento anterior, até alcançar o primeiro elemento da lista, que não tem elemento anterior (o ponteiro do elemento anterior vale NULL).



Para exemplificar a implementação de listas duplamente encadeadas, vamos novamente considerar um exemplo simples no qual queremos armazenar valores inteiros na lista. O nó da lista pode ser representado pela estrutura abaixo e a lista pode ser representada através do ponteiro para o primeiro nó.

```
1   struct lista2
2   {
3       int info;
4       struct lista2 *ant;
5       struct lista2 *prox;
6   };
7   typedef struct lista2 Lista2;
8
```

Função de inserção

O código a seguir mostra uma possível implementação da função que insere novos elementos no início da lista. Após a alocação do novo elemento, a função acerta o duplo encadeamento.

```c
/* inserção no início - Recebe como parametro a lista e o valor inteiro a ser inserido no
início da lista */

Lista2 *insere(Lista2 *l, int v)
{
    /* Cria um novo elemento na lista com */
    Lista2 *novo = (Lista2 *)malloc(sizeof(Lista2));
    /* Preenche o campo info do novo elemento criado com o valor inteiro */
    novo->info = v;
    /* O novo elemento aponta para o inicio da lista. Lembre que a lista na verdade é um
    ponteiro para o primeiro elemento, assim o novo elemento ao ser incluido no
    inicio   da lista deve apontar para o proximo que é o inicio da lista, ou seja,
    aquele que era o primeiro elemento da lista anteriormente */

    novo->prox = l;
    /* Como o novo elemento é inserido no inicio da lista, não há elemento anterior. Por
    isso, armazena a constante NULL no ponteiro que aponta para o elemento anterior*/

    novo->ant = NULL
                /* verifica se lista não está vazia */

                if (l != NULL)
                    l->ant = novo;

    return novo;
}
```

Nessa função, o novo elemento é inserido no início da lista. Assim, ele tem como próximo elemento o antigo primeiro elemento da lista e como anterior o valor NULL. A seguir, a função testa se a lista não era vazia, pois, neste caso, o elemento anterior do então primeiro elemento passa a ser o novo elemento. De qualquer forma, o novo elemento passa a ser o primeiro da lista, e deve ser retornado como valor da lista atualizada.

Função de busca

A função de busca recebe a informação referente ao elemento que queremos buscar etem como valor de retorno o ponteiro do elemento da lista que contém o valor que queremos buscar. Caso o elemento não seja encontrado na lista, o valor retornado é NULL.

```c
/* função busca: busca um elemento na lista */

Lista2 *busca(Lista2 *l, int v)
{
    Lista2 *p;
    for (p = l; p != NULL; p = p->prox)
        if (p->info == v)
            return p;

    return NULL;

    /* não achou o elemento */
}

// Pode ser feita assim também :

Lista2 *
busca(Lista2 *lista, int v)
{
    Lista2 *aux;
    if (!lista_vazia(lista))
    {
        aux = lista;
        while (aux != (Lista2 *)NULL)
        {
            if (aux->info == v)
            { // Se o campo valor do elemento for igual a busca entao  retorna o elemento
                return aux;
            }
            /* Atualiza aux com o ponteiro para o proximo elemento da lista.  */
            aux = aux->prox;
        }
        /* Senao encontrou retorna nulo */
        return NULL;
    }
}
else
{
    printf("\n A lista esta Vazia"); // Se a lista nao exister preenchida exibe mensagem informando lista vazia
    getch();
}
return NULL;

```

Função que retira um elemento da lista

A função de remoção fica mais complicada, pois temos que acertar o encadeamento duplo. Em contrapartida, podemos retirar um elemento da lista conhecendo apenas o ponteiro para esse elemento. Desta forma, podemos usar a função de busca acima para localizar o elemento e em seguida acertar o encadeamento, liberando o elemento ao final.

Se p representa o ponteiro do elemento que desejamos retirar, para acertar o encadeamento devemos conceitualmente fazer:

p->ant->prox = p->prox;

p->prox->ant = p->ant;

Isto é, o anterior passa a apontar para o próximo e o próximo passa a apontar para o anterior. Quando p apontar para um elemento no meio da lista, as duas atribuições acima são suficientes para efetivamente acertar o encadeamento da lista. No entanto, se p for um

elemento no extremo da lista, devemos considerar as condições de contorno. Se p for o primeiro, não podemos escrever p->ant->prox, pois p->ant é NULL; além disso, temos que atualizar o valor da lista, pois o primeiro elemento será removido.

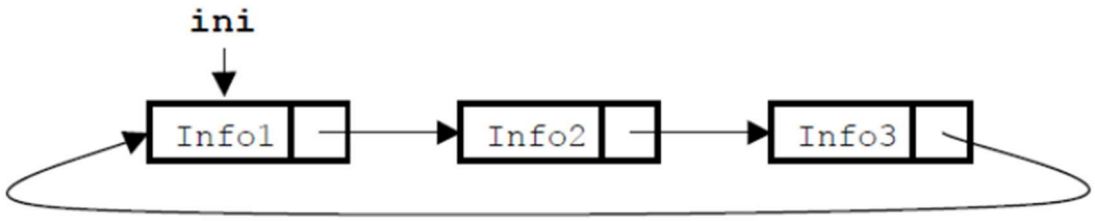
Uma implementação da função para retirar um elemento é mostrada a seguir:

```c
/* função retira: retira elemento da lista */
Lista2 *retira(Lista2 *l, int v)
{
    /* Busca o elemento a ser removido da lista */
    Lista2 *p = busca(l, v);
    /* se não achou o elemento: retorna lista inalterada */
    if (p == NULL)
        return l;

    /* se o elemento a ser retirado da lista for o primeiro elemento da lista entao o
    primeiro elemento da lista sera o proximo elemento */
    if (l == p)
        p->prox->ant = NULL;
    l = p->prox;
    else
        /* se o elemento a ser retirado da lista nao for o primeiro da lista entao o
        elemento anterior deve apontar para o elemento que esta sendo apontado pelo
        elemento a ser retirado */
        p->ant->prox = p->prox;

    /* Se o elemento a ser retirado nao for o ultimo, entao o proximo elemento deve
    apontar para o elemento anterior ao elemento a ser retirado */
    if (p->prox != NULL)
        p->prox->ant = p->ant;

    /* Efetivamente remove o elemento da memoria. Note que o elemento já não faz mais
    parte da lista pois nao e mais referenciado na lista */
    free(p);

    /* Retorna a lista atualizada */
    return l;
}
```

Listas Circulares

O que tem de novo?

O que difere uma lista circular, tanto da simplesmente encadeada ou da duplamente encadeada é que o ponteiro do ultimo elemento não aponta para NULL, e sim para o primeiro elemento da lista. A lista ela pode ser representada por um ponteiro para o primeiro elemento da lista, como na figura abaixo.



Como percorrer a lista circular?

Começando pelo elemento que o ponteiro 'ini' está apontando, percorremos ela até encontrar este mesmo elemento novamente, significando assim que foram percorridos todos os elementos da lista.

Quando usar uma lista circular do lugar de encadeada?

Existem algumas aplicações do nosso cotidiano que faz sentido relacionar com listas circulares, como os dias da semana ou os meses do ano. Ou por exemplo num jogo, quando existe uma ordem cíclica de turnos a serem jogados, na verdade tudo que você conseguir pensar em algo que for um ciclo, geralmente vai ser mais pratico a implementação de uma lista circular.

FILA

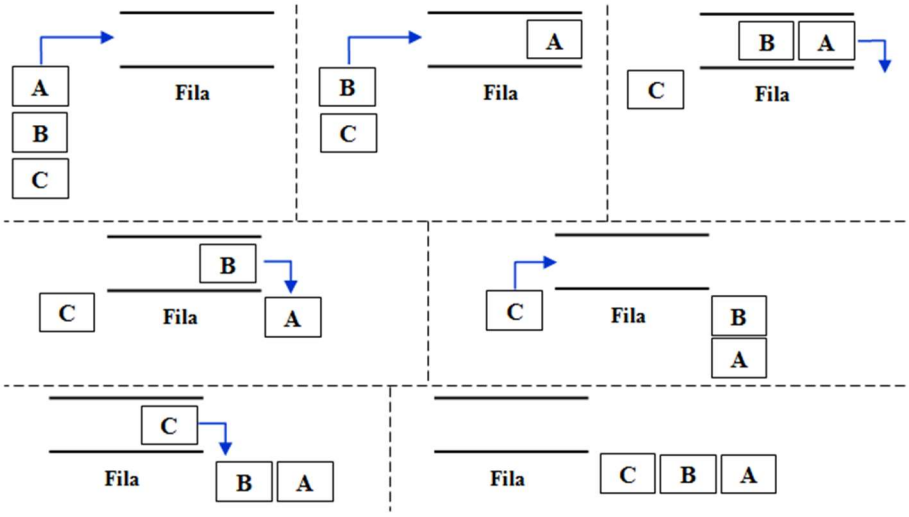
O que é?

É uma lista na qual as inserções são feitas em uma extremidade chamada "cauda" ou "fundo", e as remoções são feitas na outra extremidade, chamada "cabeça" ou "frente". Em uma fila, o primeiro a entrar é o primeiro a sair. Esta política de acesso é denominada FIFO ("First In, First Out").

- Ao inserir um nó, ele vai para a última posição da estrutura.

- Ao retirar um nó, é tirado o primeiro elemento da estrutura.

Resumindo: inserimos ao fim, e retiramos do começo.



A tabela abaixo descreve as principais operações:

Operação	Descrição
Inicializar	Cria uma fila vazia.
Empilhar	Insere um elemento no fundo (cauda) da fila.
Desempilhar	Retira um elemento que está na frente (cabeça) da fila.
Frente	Retorna o elemento que está na frente (cabeça) da fila.
Vazia	Indica se a fila está vazia.

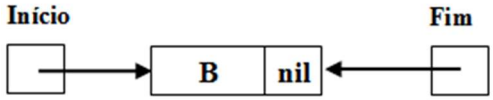
Há basicamente 2 formas de implementar uma fila:

1. **Usando uma lista encadeada**:
    1. O elemento inserido é colocado no fim da lista (cauda).
    2. A retirada é feita no início (cabeça).
2. **Usando um array**:
    1. Há duas variáveis (Início e Fim) que indicam as extremidades da fila (cabeça e cauda).
    2. Ao retirar um elemento, isto é feito na posição Início.
    3. Ao inserir, o novo elemento é colocado na posição Fim + 1.

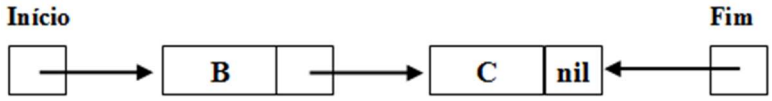
Fila baseada em lista encadeada:

Inserindo A e B:

Retirando o elemento da frente (A):



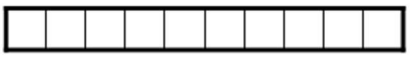
Inserindo C:



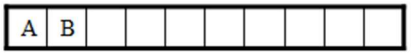
Fila baseada em array:

A implementação de uma fila é semelhante à de uma lista sequencial onde as inserções são feitas no final da lista e as remoções no início**.**

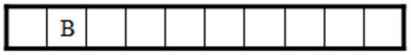
Fila Vazia:



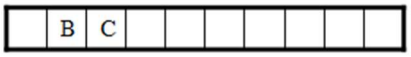
Inserindo A e B:



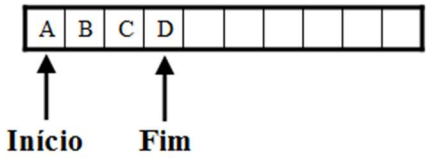
Retirando o elemento da frente (A):



Inserindo C:



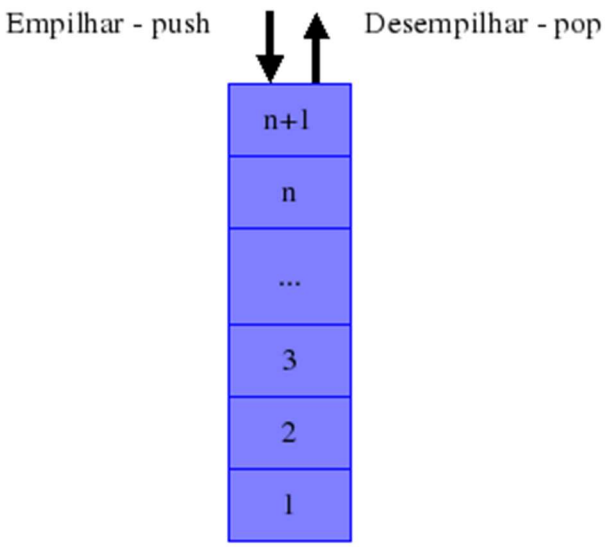
Graficamente teríamos:



PILHA

O que é?

A pilha é uma estrutura de dados que armazena os dados na ordem LIFO (Last In First Out - Último a Entrar, Primeiro a Sair). A recuperação dos dados é feita na ordem inversa da sua inserção. A ideia por trás da estrutura de dados do tipo pilha é simples: o último elemento a entrar, é sempre o primeiro a sair. Em resumo isso significa que os valores descartados serão sempre os mais recentes a terem entrado na pilha. Assim, uma pilha permite acesso a apenas um item de dados - o último inserido. Para processar o penúltimo item inserido, deve-se remover o último.



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

Em uma pilha, a manipulação dos elementos é realizada em apenas uma das extremidades, chamada de topo, em oposição a outra extremidade, chamada de base.

As operações de armazenamento e retirada são conhecidas, respectivamente, como **push( )** & **pop( ).**

Exemplos de uso de pilha em um sistema:

- Funções recursivas em compiladores;
- Mecanismo de desfazer/refazer dos editores de texto;
- Navegação entre páginas Web;
-

Operações com Pilha:

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:

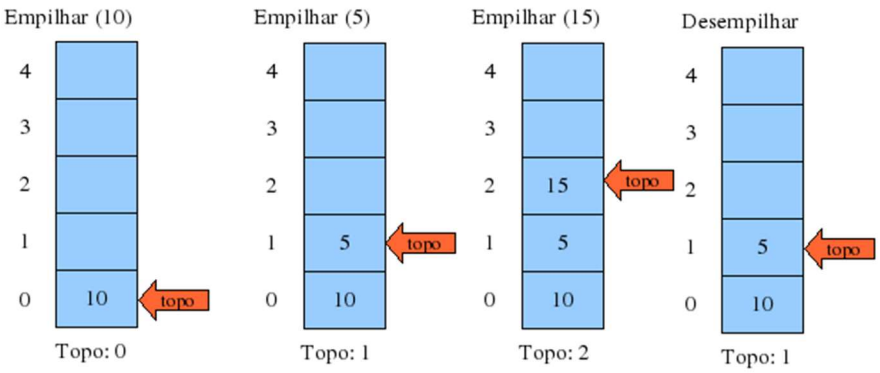
- Criação da pilha (informar a capacidade no caso de implementação sequencial - vetor);

- 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).

Supondo uma pilha com capacidade para 5 elementos (5 nós):

Inserção:

O primeiro elemento da lista representa o topo da pilha. Cada novo elemento é inserido no início da lista.



Eliminar um elemento da pilha:

Para excluir o elemento da pilha, basta excluir o elemento para o qual aponta o ponteiro início. Essa operação não permite recuperar o dado no topo da pilha, mas apenas removê-lo. A remoção de um elemento da pilha é realizada apenas alterando-se a informação da posição do topo.