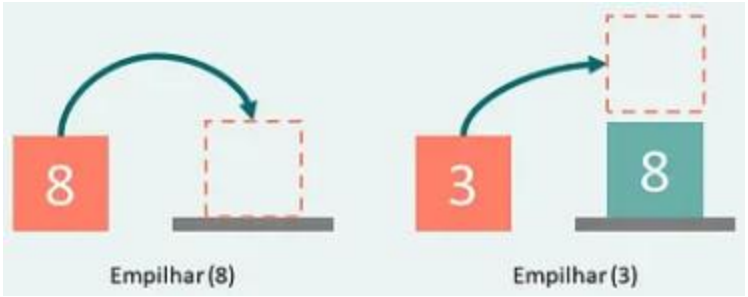


ESTRUTURA DE DADOS DINÂMICAS

Prof. Dr. Fabio Fernando Kobs

Estruturas Dinâmicas - Agenda

- Características
- Memória contígua e não contígua
- Nós
- Pilhas
- Filas
- Listas

Estruturas Dinâmicas - Características

- Assim como as estruturas estáticas, as estruturas dinâmicas ou estruturas ligadas representam sequências lineares dos itens. Porém, não se pode acessar imediatamente um item especificando a posição de índice.
- As estruturas dinâmicas são utilizadas para relacionar itens que precisam ser manipulados em tempo de execução com dimensão indefinida.
- O acesso em uma estrutura dinâmica deve começar em uma extremidade da estrutura e seguir as ligações até que a posição (ou item) desejada seja alcançada.

Estruturas Dinâmicas - Características

A maneira como a memória é alocada para estruturas ligadas também é bastante diferente daquela dos *arrays* (vetores) e há duas consequências importantes para as operações de inserção e remoção:

- Depois que um ponto de inserção ou remoção foi encontrado, a inserção ou remoção pode ocorrer sem nenhum deslocamento de itens de dados na memória.
- A estrutura ligada pode ser redimensionada durante cada inserção ou remoção sem custo extra de memória e sem cópia de itens de dados.

A principal vantagem da estrutura dinâmica em relação à estática não é o desempenho no tempo, mas o desempenho da memória.

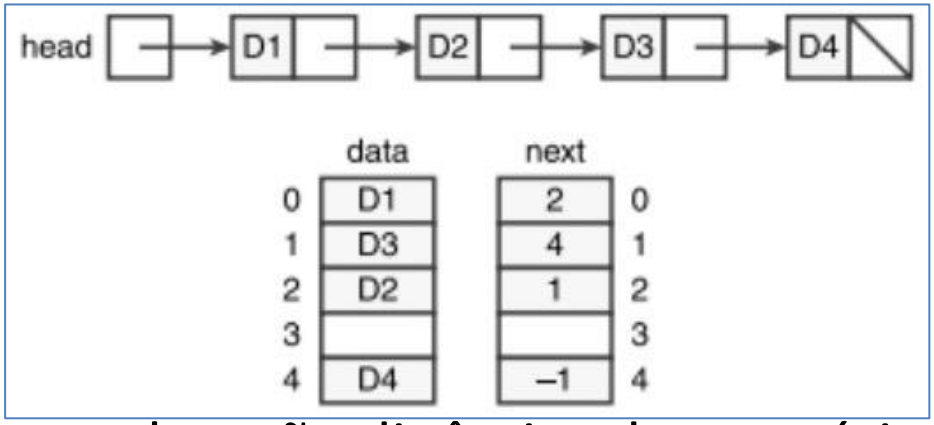
Estruturas Dinâmicas – Memória e nós

- Os itens do vetor devem ser armazenados na memória contígua. Isso significa que a sequência lógica dos itens no vetor está acoplada a uma sequência física das células na memória.
- Já uma estrutura ligada desacopla a sequência lógica dos itens na estrutura de qualquer ordem na memória. Isto é, a célula de determinado item em uma estrutura ligada pode ser encontrada em qualquer lugar da memória, desde que o computador possa seguir uma ligação para o endereço ou a localização. Esse tipo de representação de memória chama-se memória não contígua.
- A unidade básica da representação em uma estrutura dinâmica é um nó. Um nó individualmente ligado contém os atributos:
  - Um item de dados;
  - Uma ligação para o próximo nó na estrutura.
- No caso de um nó duplamente ligado, contém um atributo para o nó anterior na estrutura.

Estruturas Dinâmicas – Nós no Python

Em Python configuram-se nós e estruturas ligadas por meio de referências a objetos.

Qualquer variável pode se referir a qualquer coisa, incluindo o valor *None*, o que pode significar uma ligação vazia. Portanto, configura-se um nó individualmente ligado definindo um objeto que contém dois campos: uma referência a um item de dados e uma referência a outro nó. Exemplo de uma estrutura ligada:



O Python fornece alocação dinâmica de memória não contígua para cada novo objeto de nó, bem como retorno automático dessa memória ao sistema (coleta de lixo) quando o objeto não pode mais ser referenciado pelo aplicativo.

Estruturas Dinâmicas – Nós no Python

Para definir uma classe de nó individualmente ligado, as variáveis de instância de um objeto de nó são geralmente referenciadas sem chamadas de método, e os construtores permitem ao usuário definir a(s) ligação(ões) de um nó quando este é criado.

Na sequência o código para uma classe de nó simples e individualmente ligado:

```
class Node:     # Representa um nó individualmente ligado.

        def __init__(self, dado, proximo = None):
                self.dado = dado
                self.proximo = proximo
```

Estruturas Dinâmicas - Exemplo

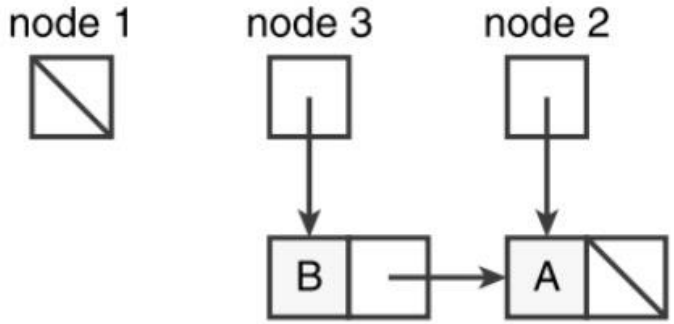
- Exemplo usando a classe Node (*slide* anterior):

Variáveis de nó são inicializadas para o valor None ou um novo objeto Node. O próximo segmento de código mostra algumas variações dessas duas opções:

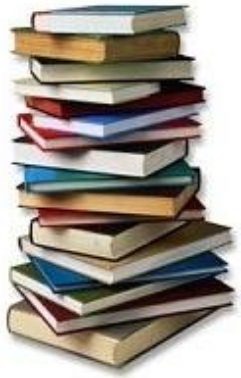
```
# Apenas uma ligação vazia
node1 = None
# Um nó contendo dados e ligação vazia
node2 = Node("A", None)
# Um nó contendo dados e ligação para o node2
node3 = Node("B", node2)
```

A Figura abaixo mostra o estado das três variáveis após a execução desse código:

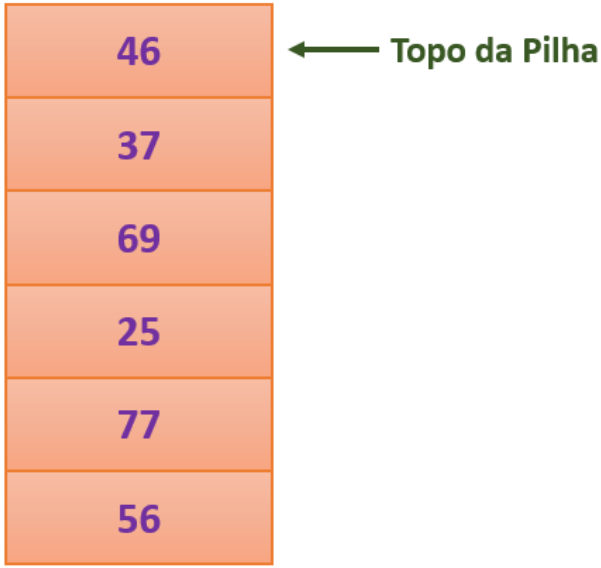
- node1 não aponta para nenhum objeto de nó (é *None*).
- node2 e node3 apontam para objetos que estão ligados.
- node2 aponta para um objeto cujo próximo ponteiro ou referência é *None*.



PILHAS - Definição



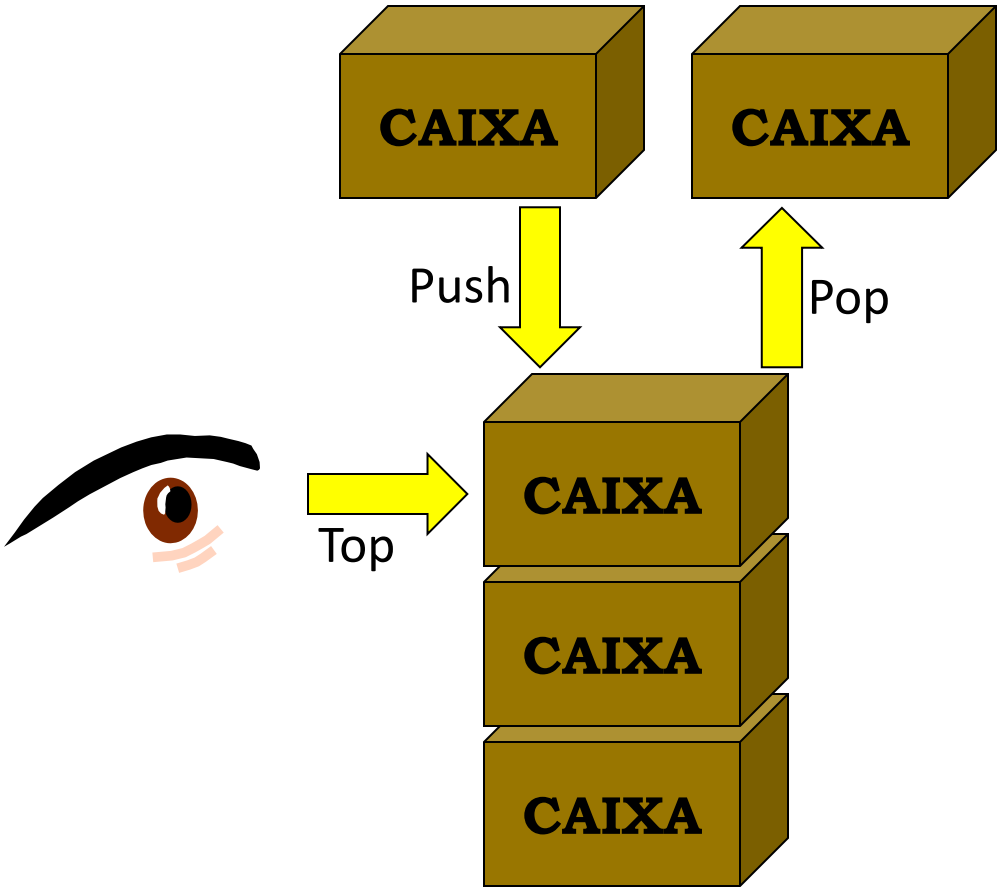
- Pilhas são coleções lineares em que o acesso é completamente restrito a apenas uma extremidade, chamada topo.
- Ou seja, os novos itens só podem ser adicionados no topo da pilha e também só podem ser removidos os itens do seu topo.
- Conhecida como uma estrutura LIFO *(last-in, first-out*, ou último a entrar é o primeiro a sair, UEPS)



PILHAS

- Operações básicas:

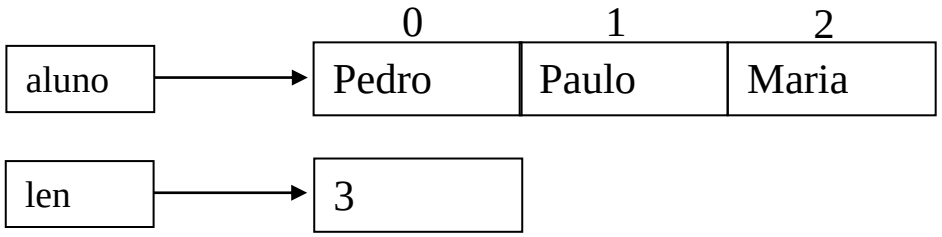
- Empilhar (*Push*)
- Desempilhar (*Pop*)
- Ler o topo sem desempilhar (*Top*)



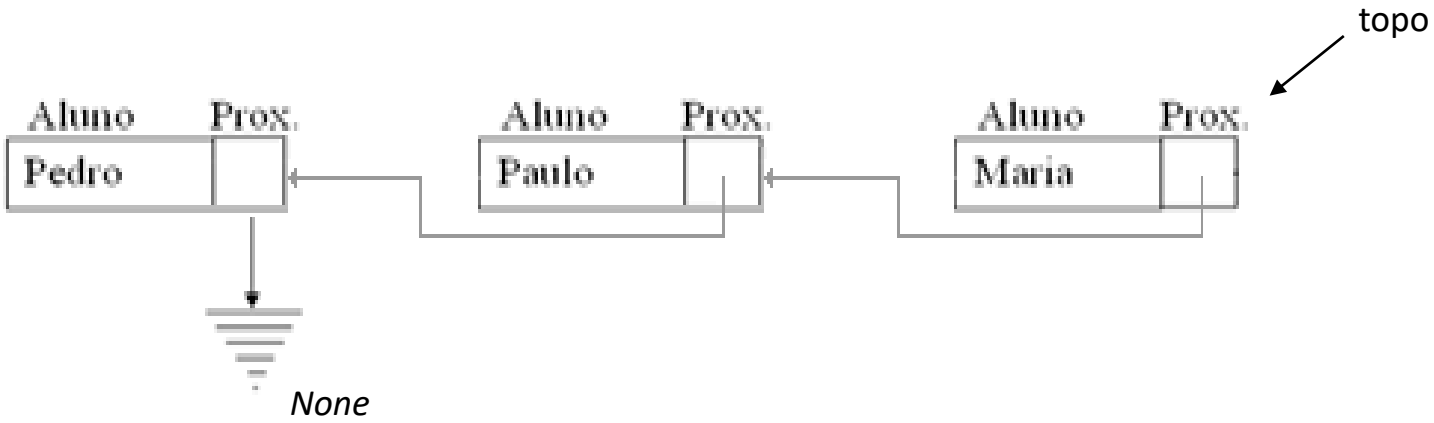
PILHAS

Comparações entre os tipos de pilhas

- **Pilha estática (sequencial)**:



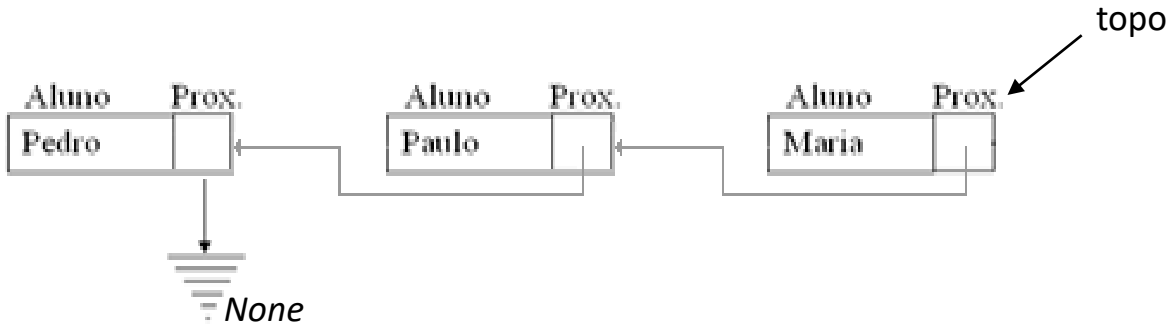
- **Pilha dinâmica ou ligada (encadeada)**:



PILHAS

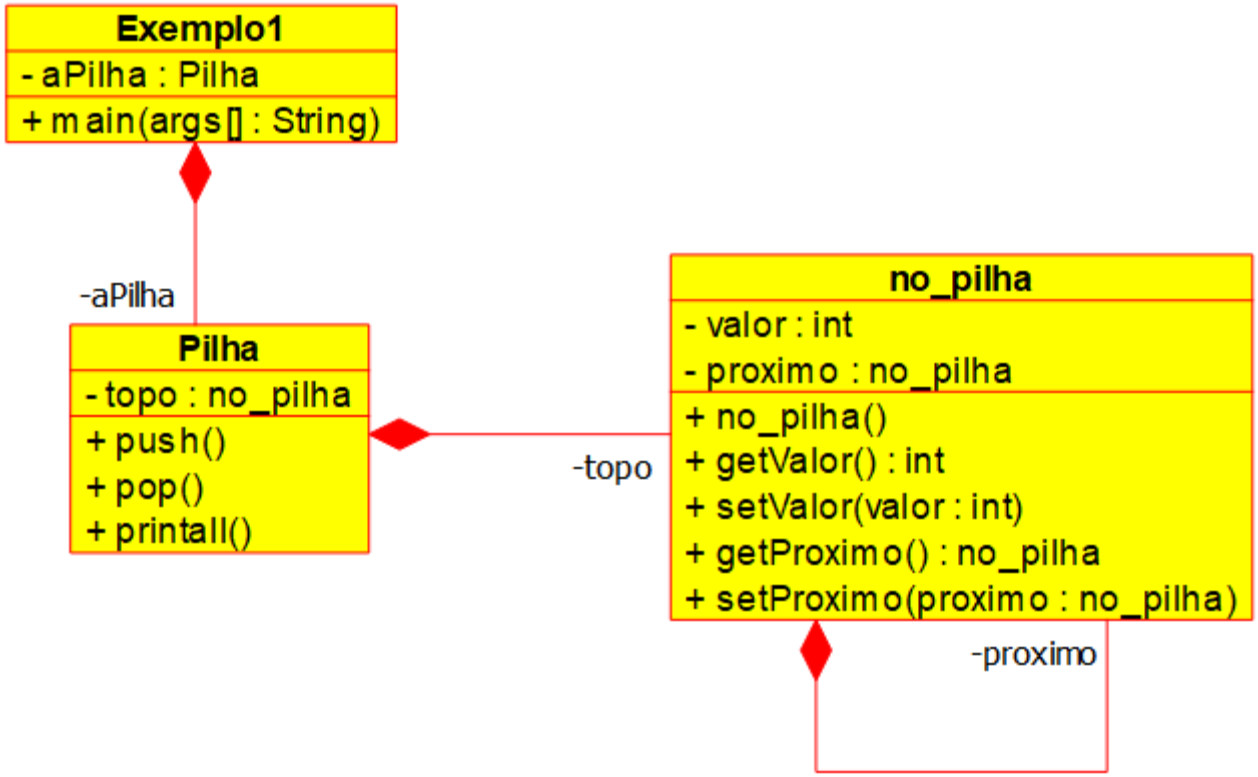
Implementação

- A referência inicial ou topo, possuirá sempre referência para a extremidade da pilha, ou seja, para o último objeto empilhado.
- No início, enquanto pilha vazia topo conterá ***None***.
- Observando o exemplo a seguir de uma pilha encadeada, a variável **topo** teria como referência o objeto com o nome “Maria”.



PILHAS
EXEMPLO

- Aplicação de pilha encadeada na leitura de n elementos inteiros.



PILHAS - **Exemplo**

```
aPilha = Pilha()
while True:
    op = int(input("\n1 - Empilhar\n2 -
     Desempilhar\n3 - Imprime a
     pilha\n4 - Sair\n\nOpção: "))
    if op == 1:
        aPilha.push()
    elif op == 2:
        aPilha.pop()
    elif op == 3:
        aPilha.printall()
    else:
        break
```

PILHAS - **Exemplo**

```
class No_pilha:

    def __init__(self):
        self.__valor = 0
        self.__proximo = None

    def getValor(self):
        return self.__valor
    def setValor(self, valor):
        self.__valor = valor

    def getProximo(self):
        return self.__proximo
    def setProximo(self, prox):
        self.__proximo = prox
```

PILHAS - **Exemplo**

```
class Pilha:
    def __init__(self):
        self.__topo = None

    def push(self):
        temp_no = No_pilha()
        if temp_no:  # not None
            temp_no.setValor(int(input("Entre com um valor: ")))
            temp_no.setProximo(self.__topo);
            self.__topo = temp_no

    def pop(self):
        if self.__topo:  # not None
            print("\nValor a remover: " , self.__topo.getValor())
            self.__topo = self.__topo.getProximo()
        else:
            print("Pilha vazia...")

    def printall(self):
```

PILHAS - **Exemplo**

```
def printall(self):
        if not self.__topo:  # None
            print("Pilha vazia...")
            return
        temp_no = self.__topo
        saida = "Pilha:\n"
        while temp_no:   # not None
             saida += str(temp_no.getValor()) + "\n"
             temp_no = temp_no.getProximo()
        print(saida)
```

PILHAS ENCADEADAS – **EXERCÍCIOS**

01 a) Incluir no Exemplo 1, a função para imprimir o elemento topo da pilha.

01 b) Incluir no Exemplo 1, a função para imprimir a soma dos elementos da pilha.

01 c) Incluir no Exemplo 1, a função para imprimir a média dos elementos da pilha.

02) Altere o programa da questão anterior, de forma que permita a inclusão em cada nó do nome do aluno e da sua idade. Alterar da mesma forma os demais métodos (exceto item 01 b).

03) Escreva um programa usando pilhas para determinar se uma *String* é um palíndromo.

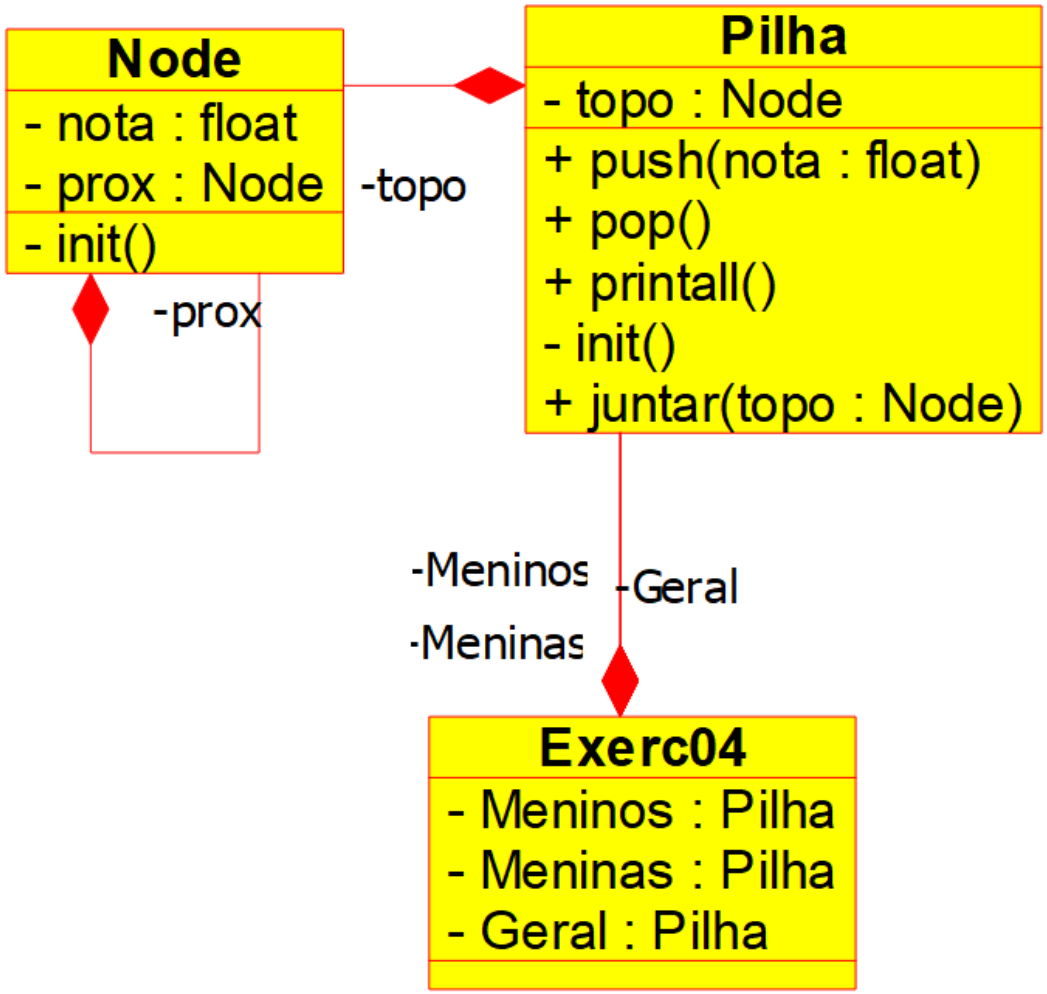
PILHAS ENCADEADAS – **EXERCÍCIOS**

04) Escreva um programa que empilhe notas quaisquer na pilha 1 e na pilha 2, observando que as notas dos meninos devem estar na pilha 1, já as notas das meninas na pilha 2. Crie uma função que concatene ambas as pilhas (gerando uma terceira pilha), contendo as notas dos meninos e das meninas. Imprimir todas as pilhas.

Bônus: incluir na impressão de cada pilha a média das suas notas.

PILHAS ENCADEADAS – **EXERCÍCIOS**

04) Diagrama de classe:



PILHAS ENCADEADAS – **EXERCÍCIOS**

05) Escreva um programa que leia uma sequência de caracteres (*string*) e imprima-a em ordem inversa. Use uma pilha encadeada para tal.

06) Escreva um programa com duas pilhas de valores inteiros. Para cada uma:

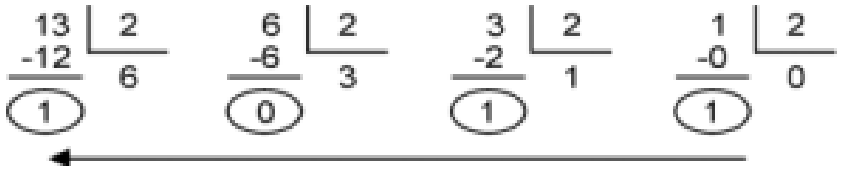
- se positivo, inserir na pilha P;
- se negativo, inserir na pilha N;
- se zero, retirar um elemento de cada pilha.

Inclua uma operação para imprimir as duas pilhas.

PILHAS ENCADEADAS – **EXERCÍCIOS**

07) Escreva um programa que leia um inteiro positivo e imprima o binário desse inteiro.

Dica: Para converter um número decimal em binário, deve-se efetuar sucessivas divisões por 2 até que um quociente 0 seja obtido. Então, os restos das divisões efetuadas, tomados na ordem inversa àquela em que foram produzidos, formam o número binário desejado. Por exemplo, para converter o número decimal 13 em binário, fazemos as seguintes divisões:



Se a cada divisão feita, o resto obtido for empilhado, no final do processo, basta retirar os valores guardados na pilha para ter o número correspondente em binário (que para este exemplo, é 1101).

Depois, tente fazer o inverso, ou seja, converter um número binário para decimal.

08) *Bônus 1*: Implemente uma pilha onde cada item seja um número variável de inteiros. Elabore operações *push*, *pop*, *printstacktop*, *printall*, *popall*.

09) *Bônus 2*: Inclua uma operação para ordenar os valores de cada item na forma crescente (questão anterior).

PILHAS ENCADEADAS – **EXERCÍCIOS**

10) *Bônus 3*: O estacionamento X contém uma única alameda que guarda até cinco carros. Existe apenas uma entrada/saída no estacionamento, em uma extremidade da alameda. Se chegar um cliente para retirar um carro que não seja o mais próximo da saída, todos os carros bloqueando seu caminho sairão do estacionamento, o carro do cliente será manobrado para fora do estacionamento, e os outros carros voltarão ao estacionamento. Escreva um programa para o estacionamento, contendo o número da placa do carro. Presume-se que os carros cheguem na ordem especificada pela entrada. Quando um carro sair do estacionamento, a mensagem deverá exibir o valor a pagar (considerar R$ 5,00 até meia hora, R$ 10,00 a primeira hora, e R$ 8,00 as demais).

PILHAS
REFERÊNCIAS

- DEITEL, H. M.; DEITEL, P. J. **Como programar em C**. 2. ed. Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora S.A., 1999. ISBN 85-216-1191-9.
- LAFORE, R. **Estruturas de dados & algoritmos em Java**. Rio de Janeiro: Editora Ciência Moderna Ltda., 2004. ISBN 85-7393-375-5.
- LAMBERT, Kenneth A. **Fundamentos de Python**: estruturas de dados. [Digite o Local da Editora]: Cengage Learning Brasil, 2022. E-book. ISBN 9786555584288. Disponível em: https://app.minhabiblioteca.com.br/#/books/9786555584288/. Acesso em: 07 mar. 2024.
- PINTO, Rafael A.; PRESTES, Lucas P.; SERPA, Matheus da S.; et al. **Estrutura de dados**. [Digite o Local da Editora]: Grupo A, 2020. E-book. ISBN 9786581492953. Disponível em: https://app.minhabiblioteca.com.br/#/books/9786581492953/. Acesso em: 06 mar. 2024.
- TENENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. **Estruturas de dados usando C**. Trad. Teresa Cristina Félix de Souza. São Paulo: Makron Books, 1995. ISBN 85-346-0348-0.