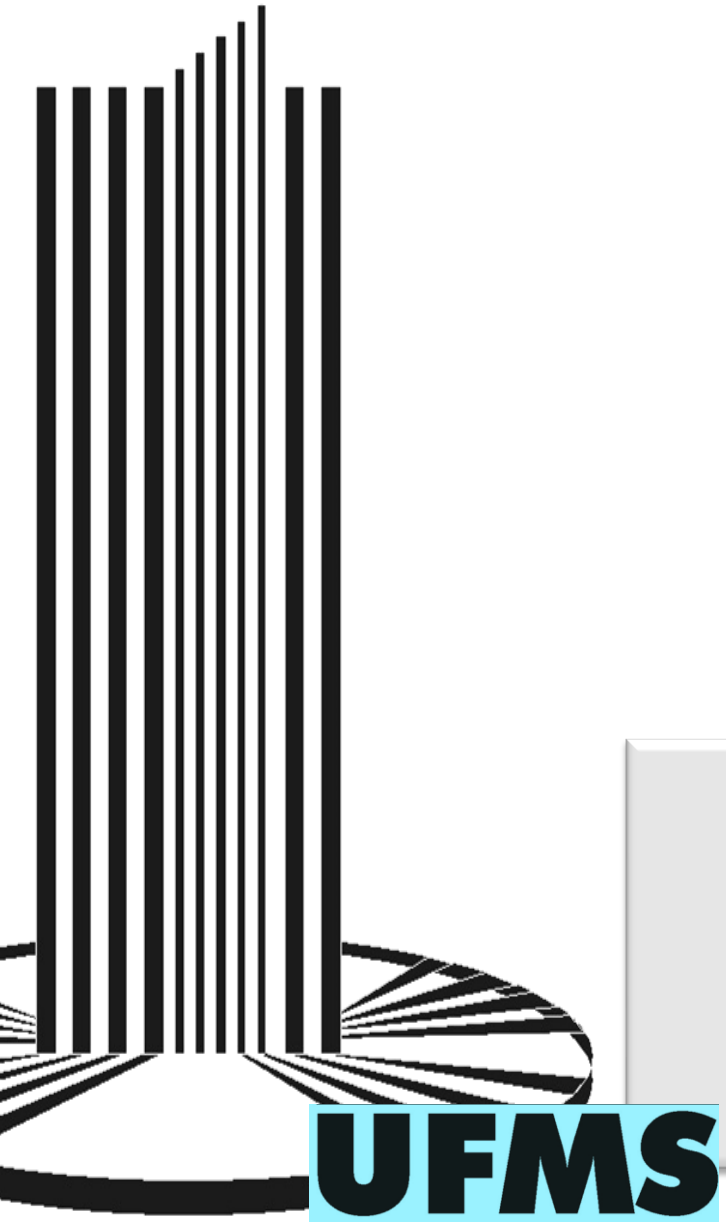
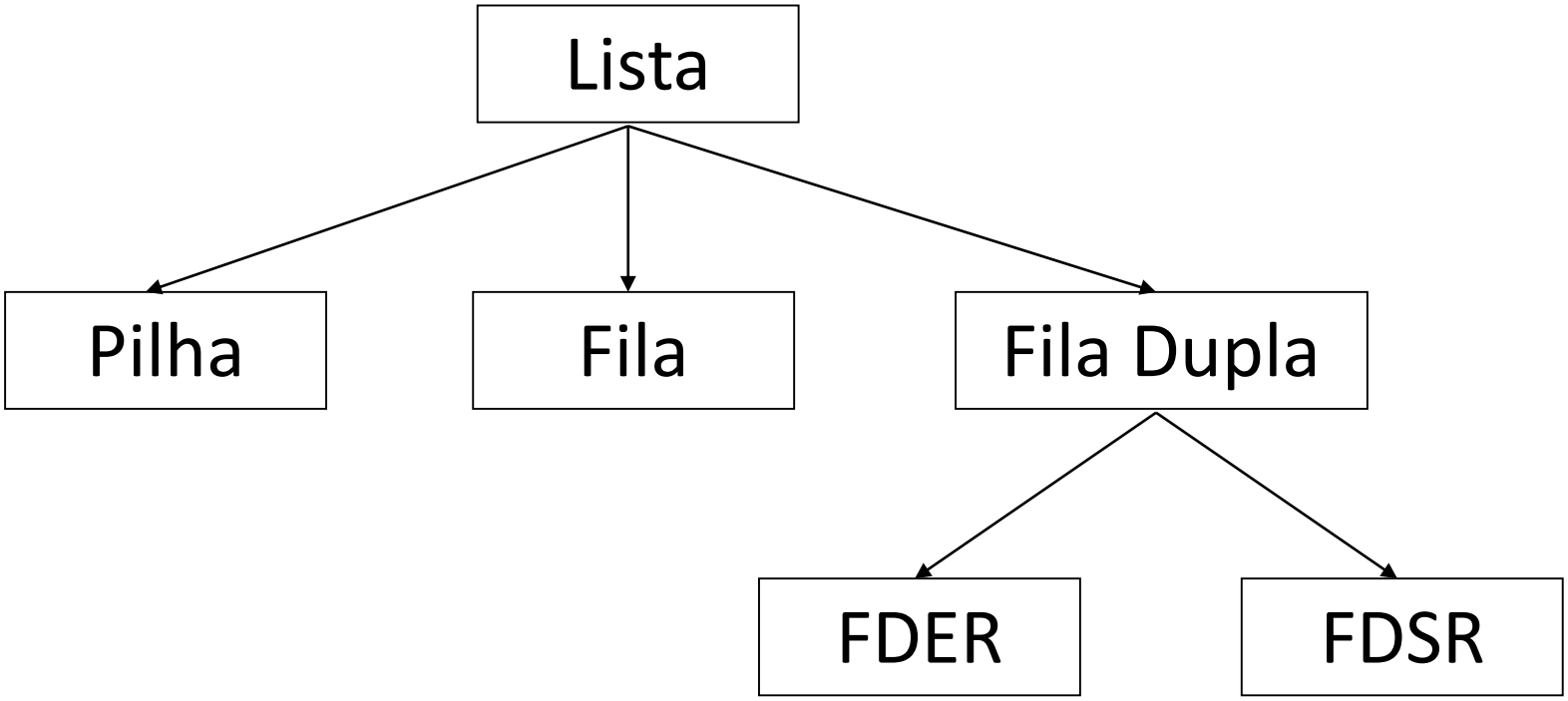


Pilha

Profª Yorah Bosse

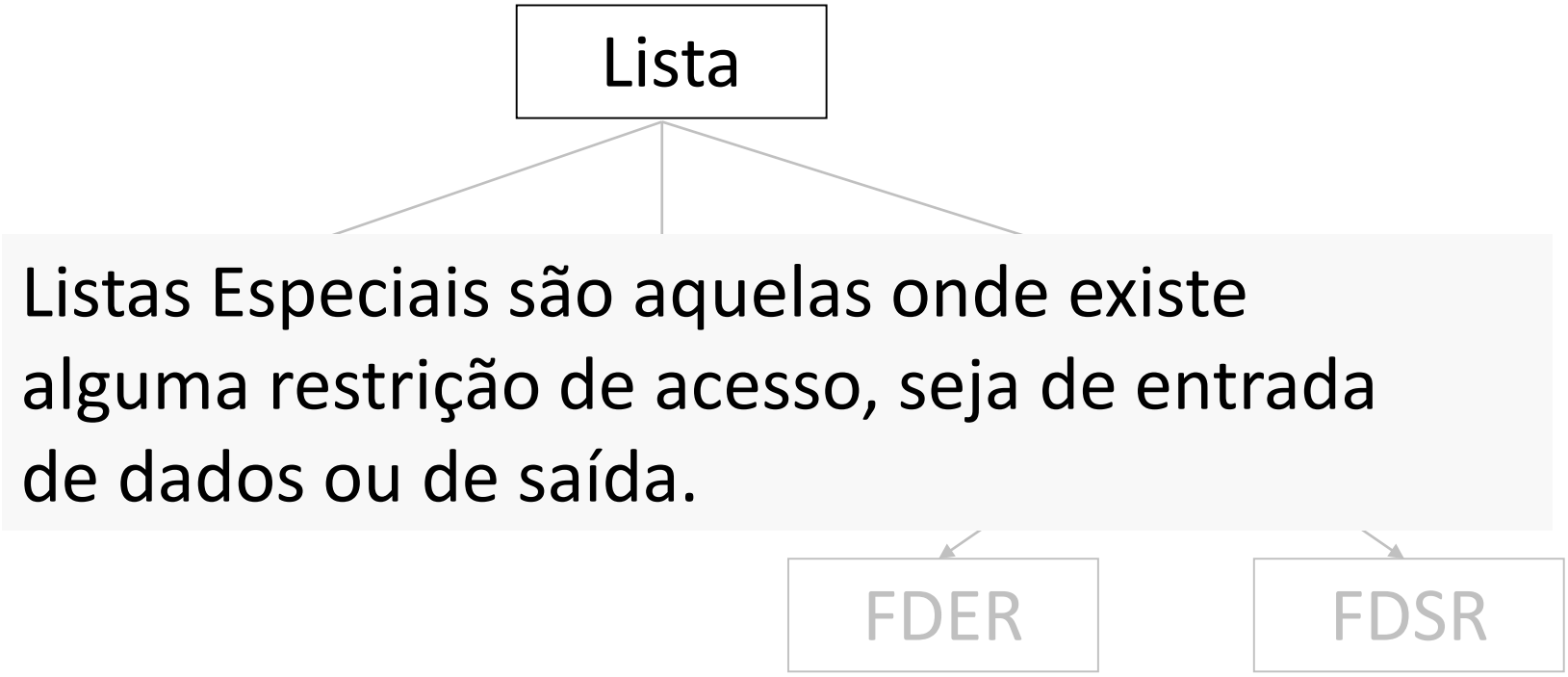
yorah.bosse@gmail.com
yorah.bosse@ufms.br

Listas Especiais



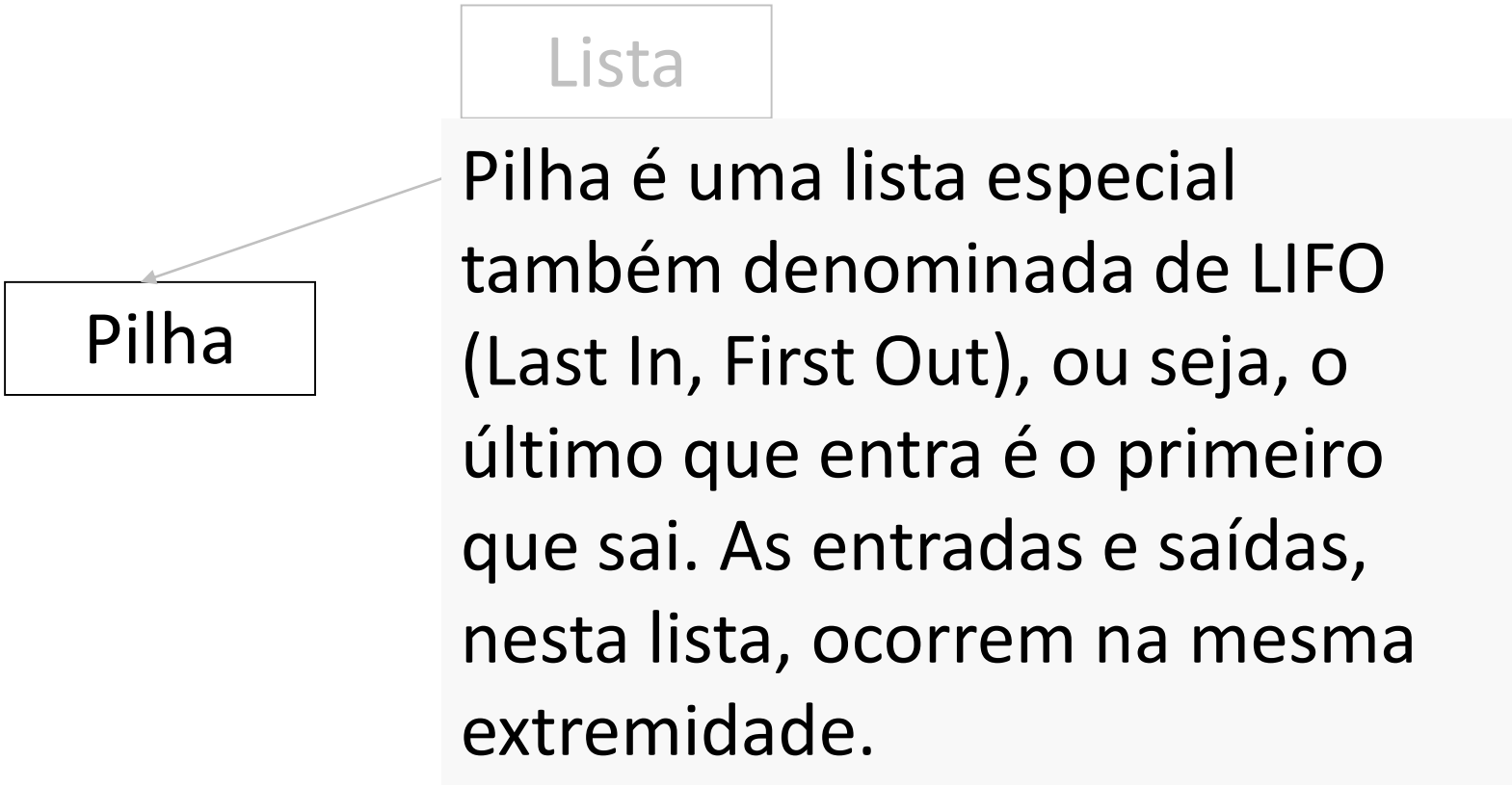
Fonte: PEREIRA, Silvio do Lago. **Estruturas de Dados Fundamentais.** São Paulo : Érica, 1996. Página 9.

Listas Especiais



Fonte: PEREIRA, Silvio do Lago. **Estruturas de Dados Fundamentais.** São Paulo : Érica, 1996. Página 9.

Listas Especiais



Fonte: PEREIRA, Silvio do Lago. **Estruturas de Dados Fundamentais.** São Paulo : Érica, 1996. Página 9.

Formas de Implementação

- **Seqüencial**: Os elementos estão fisicamente contíguos, um após o outro. Pode ser representada por um vetor na memória principal ou um arquivo sequencial em disco.

- **Encadeada**:
  - Estática: Inicio da lista é definido por uma variável inteiro que estabelece onde começa o encadeamento. Cada elemento possui um campo do tipo int, que estabelece o sucessor do mesmo.
    - Desvantagens: Quantidade máxima de elementos estabelecida (Alocação Estática).
  - Dinâmica: Início da lista é definido por um apontador que armazena o endereço do elemento inicial da lista. Cada elemento da lista possui um campo do tipo apontador que armazena o endereço do próximo elemento da lista. Cada elemento da lista é alocado dinamicamente.

Listas Lineares

- Abaixo uma relação de algumas Listas Lineares :
  - Seqüêncial
  - Estática Simples* ou duplamente Encadeada
  - Dinâmica Simples* ou duplamente* Encadeada
  - Dinâmica Simples ou duplamente* Encadeada com descritor
  - Dinâmica Circular Simples ou duplamente* Encadeada
  - Pilhas (stack) estáticas e dinâmicas* simples* ou duplamente Encadeadas
    - Pilha Dinâmica Simplesmente Encadeada
  - Filas (queue) estáticas e dinâmicas* simples* ou duplamente Encadeadas

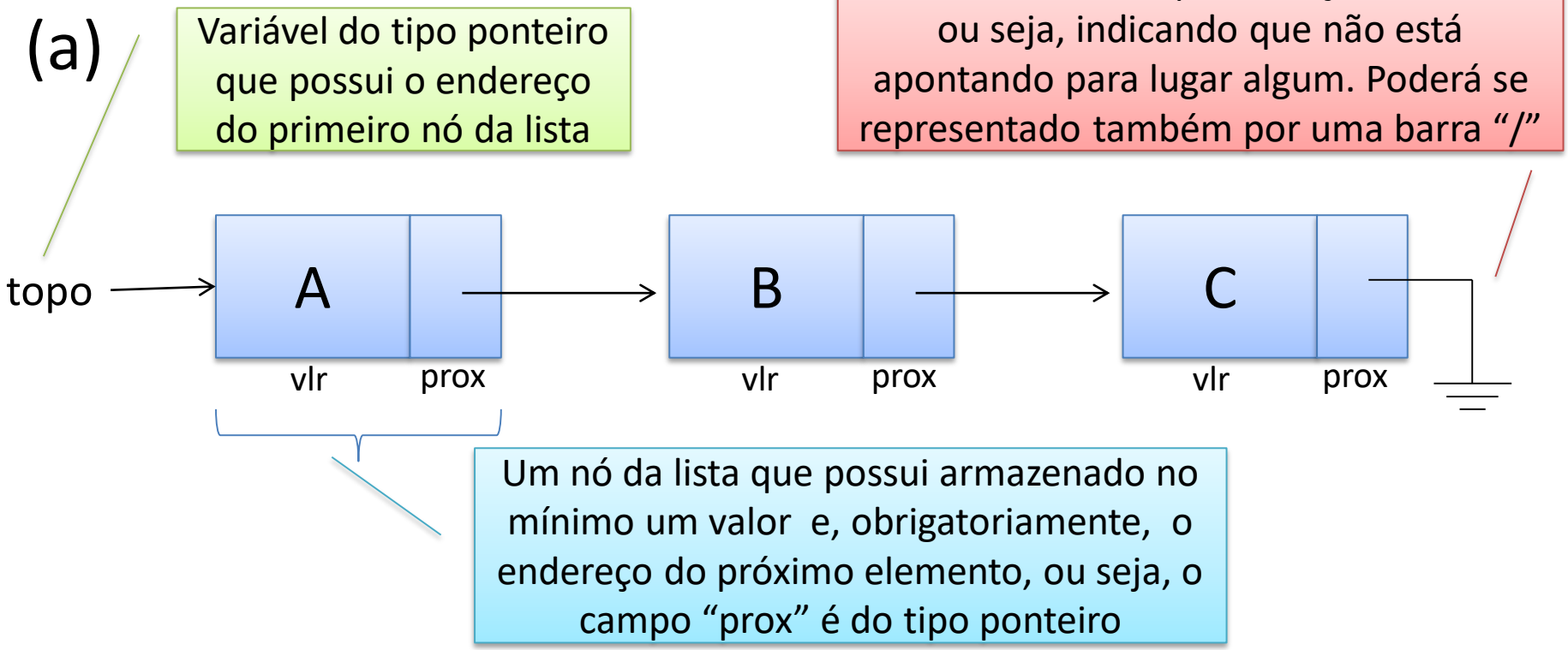
* serão estudadas nesta disciplina

- Operações possíveis:
  - Inicializar
  - Incluir no topo da pilha – Empilhar - push
  - Excluir do topo da pilha- Desempilhar - pop
  - Acessar topo sem removê-lo
  - Verificar se a pilha está Vazia

Nó

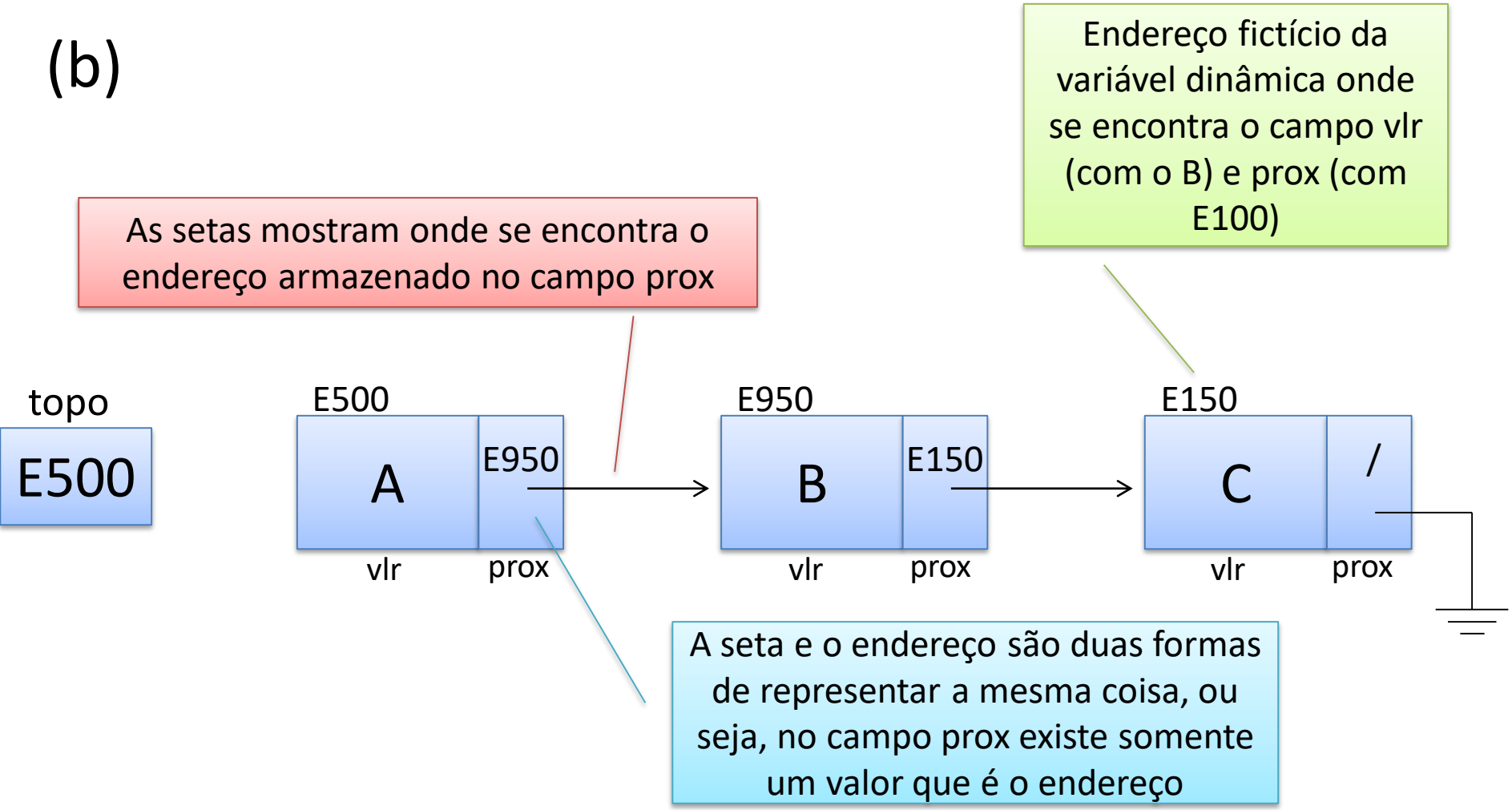
- Um nó é um espaço da lista destinado a armazenar algum valor
- A pilha é normalmente representada pelo desenho abaixo:

(a)



- Variações na representação:

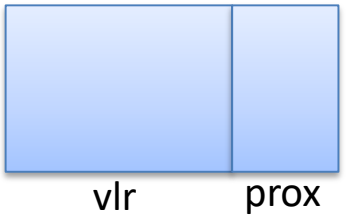
(b)



Implementação – PILHA - Criação

- A estrutura necessária para a criação do nó da lista é:

```
typedef struct dadosPilha{
        char vlr;
        struct dadosPilha *prox;
}sPILHA;
```



- Para criar a pilha é declarada uma variável do tipo ponteiro desta estrutura, que deve ser inicializada com NULL:

```
sPILHA *pilha;
pilha = inicializa(pilha);
```

```
sPILHA *inicializa(sPILHA *topo){
      topo = NULL;
      return topo;
}
```

Implementação – PILHA – Lista Vazia?

Visualização Gráfica

pilha

/

topo

/

- Quando a pilha está vazia, a variável “topo” do tipo ponteiro terá NULL armazenado dentro dela. A função que verifica esta condição da lista é:

```
int estaVazia(sPILHA *topo){
     return (topo == NULL?1:0);
}
```

Implementação – PILHA – Inserir (push)

Visualização Gráfica

pilha

/

topo

/

dado

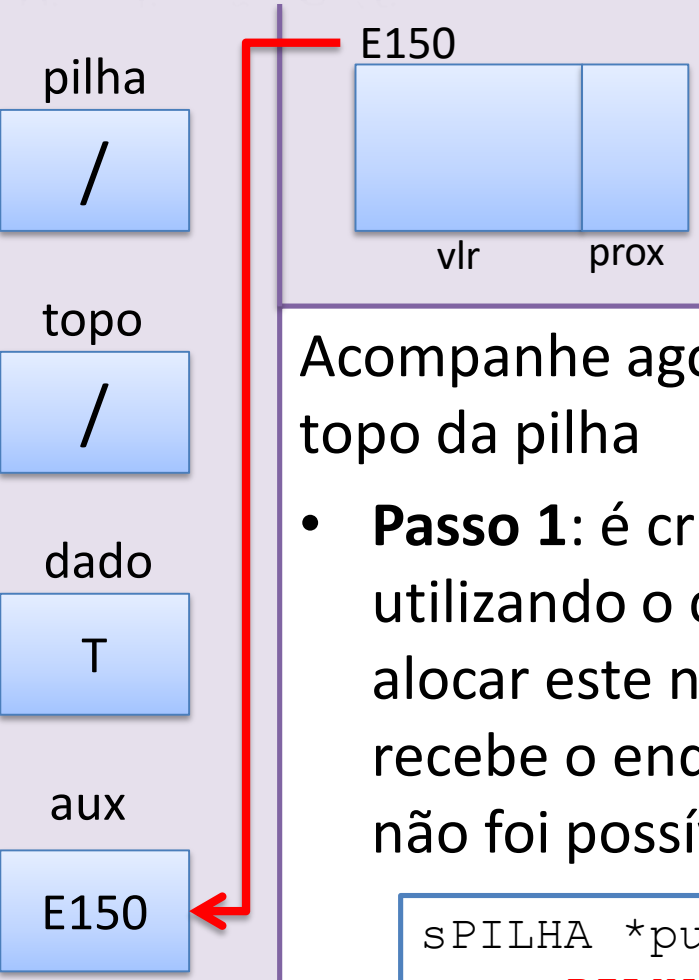
T

- Para a função que faz somente a inserção de um elemento na lista, são passados dois parâmetros:
  - a lista onde será feita a inserção
  - o dado a ser inserido
- A função retornará a lista onde a inserção foi realizada para dentro da variável “pilha”.
- O dado poderá ser inserido somente no topo da pilha.
- Veremos a seguir a forma como são inseridos os valores na pilha:

```
sPILHA *push(sPILHA *topo, char dado){...}
```

Implementação – Pilha – Inserir (push)

Visualização Gráfica



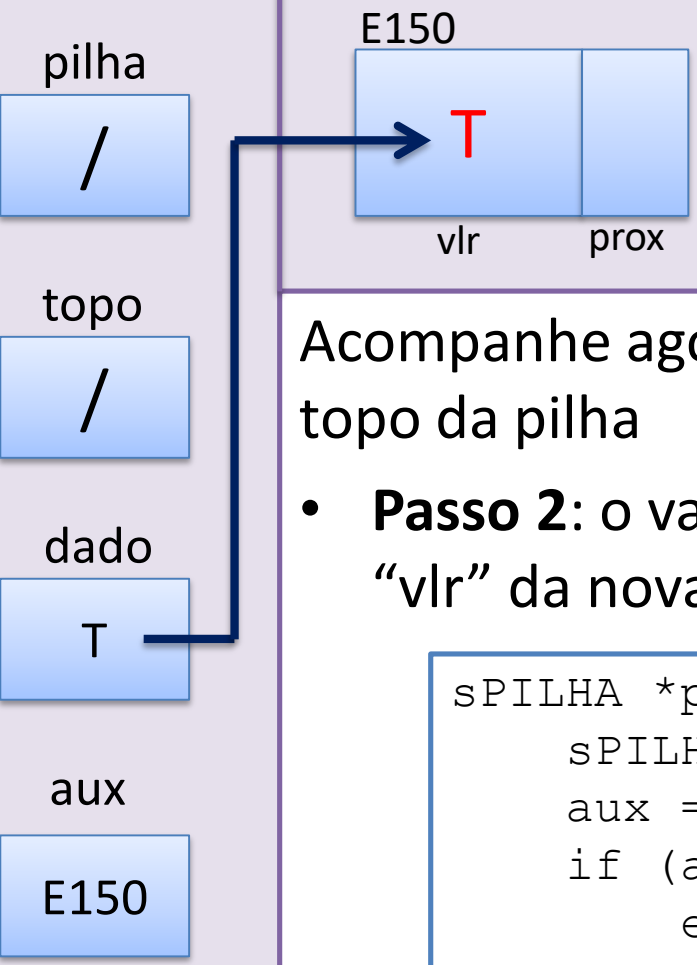
Acompanhe agora os passos para inserir um elemento no topo da pilha

- **Passo 1**: é criado um novo nó (variável dinâmica) , utilizando o comando malloc, e verificado se foi possível alocar este novo espaço na memória, ou seja, se aux, que recebe o endereço do novo espaço, tiver NULL é porque não foi possível.

```
sPILHA *push(sPILHA *topo, char dado){
     sPILHA *aux;
     aux = (sPILHA *) malloc (sizeof(sPILHA));
     if (aux == NULL)
          exit (1);
}
```

Implementação – Pilha – Inserir (push)

Visualização Gráfica



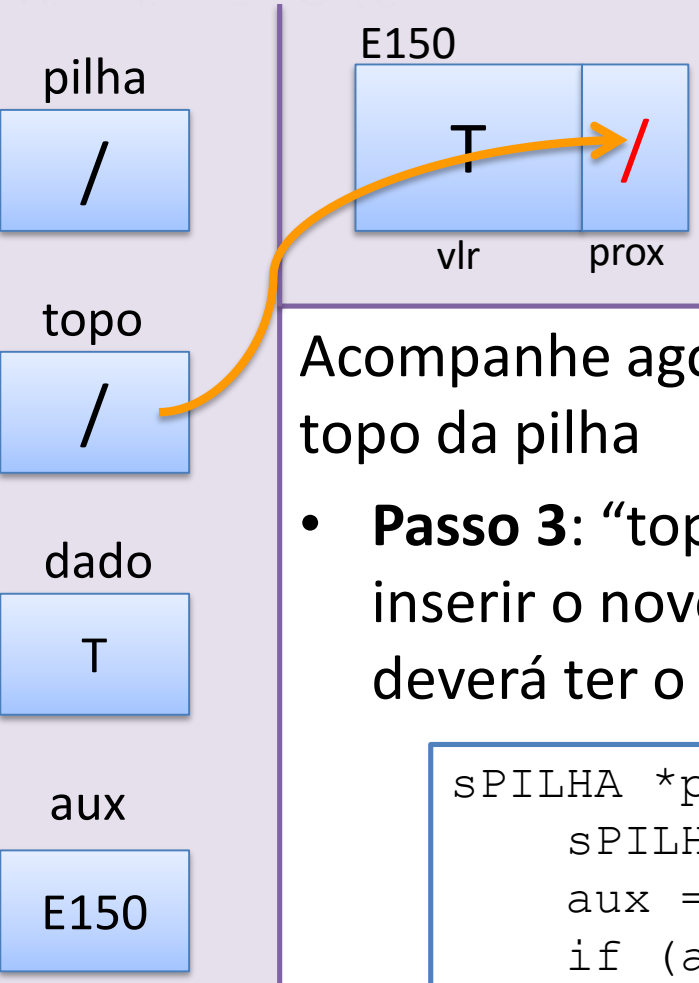
Acompanhe agora os passos para inserir um elemento no topo da pilha

- **Passo 2**: o valor da variável "dado" é inserido no campo "vlr" da nova variável dinâmica.

```
sPILHA *push(sPILHA *topo, char dado){
     sPILHA *aux;
     aux = (sPILHA *) malloc (sizeof(sPILHA));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
}
```

Implementação – Pilha – Inserir (push)

Visualização Gráfica



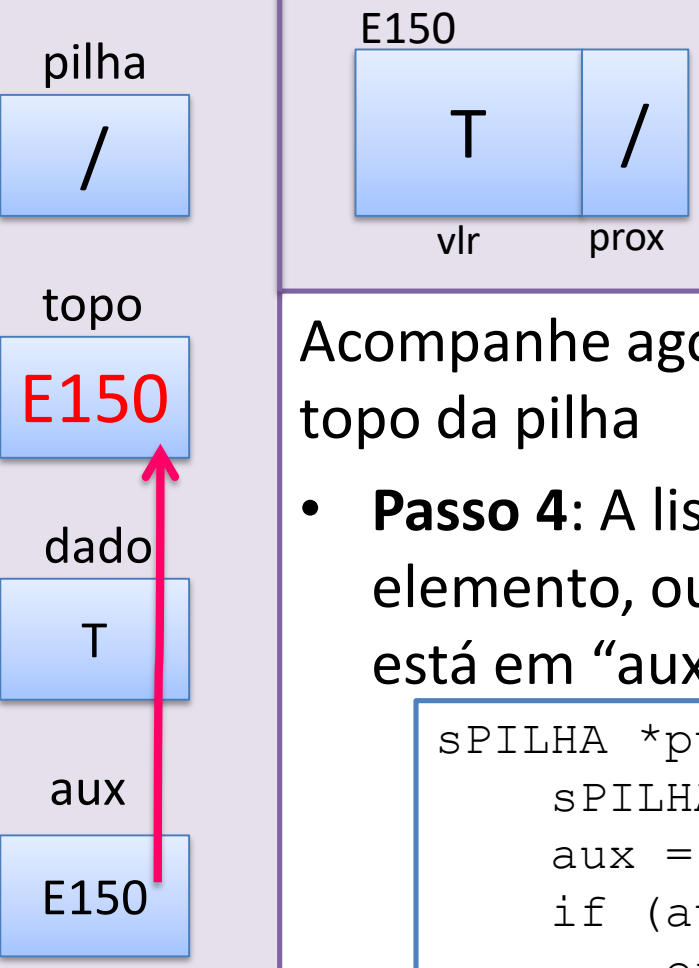
Acompanhe agora os passos para inserir um elemento no topo da pilha

- **Passo 3**: "topo" tem o endereço do topo da pilha. Ao inserir o novo elemento, o campo "prox" do novo nó deverá ter o endereço contido em "topo"

```
sPILHA *push(sPILHA *topo, char dado){
     sPILHA *aux;
     aux = (sPILHA *) malloc (sizeof(sPILHA));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
     aux->prox = topo;
}
```

Implementação – Pilha – Inserir (push)

Visualização Gráfica



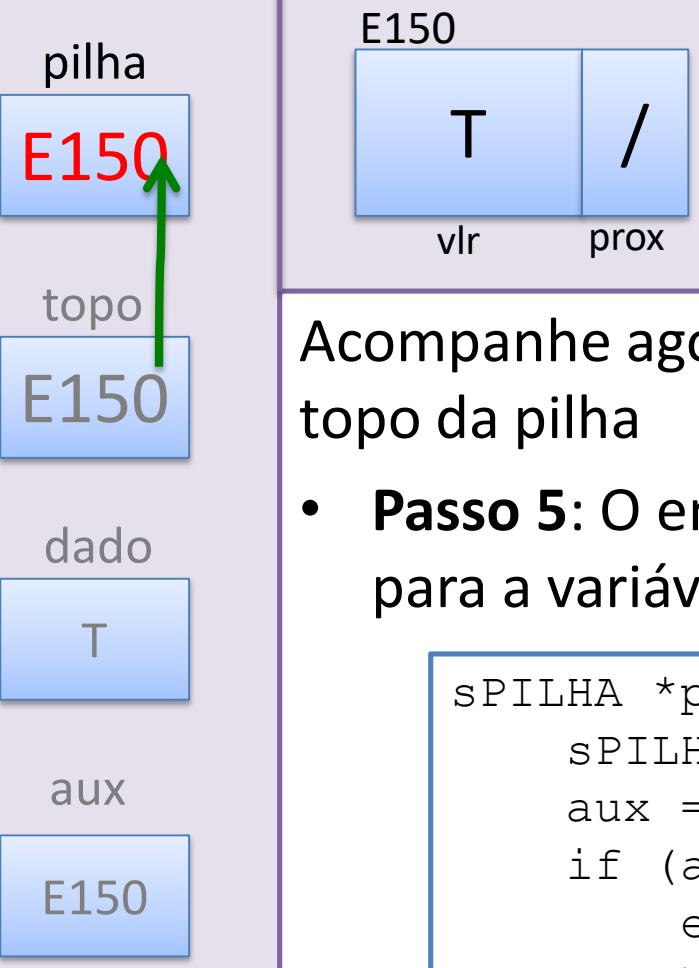
Acompanhe agora os passos para inserir um elemento no topo da pilha

- **Passo 4**: A lista "topo" precisa apontar para este novo elemento, ou seja, precisa receber o endereço dele que está em "aux"

```
sPILHA *push(sPILHA *topo, char dado){
    sPILHA *aux;
    aux = (sPILHA *) malloc (sizeof(sPILHA));
    if (aux == NULL)
        exit (1);
    aux->vlr = dado;
    aux->prox = topo;
    topo = aux;
}
```

Implementação – Pilha – Inserir (push)

Visualização Gráfica



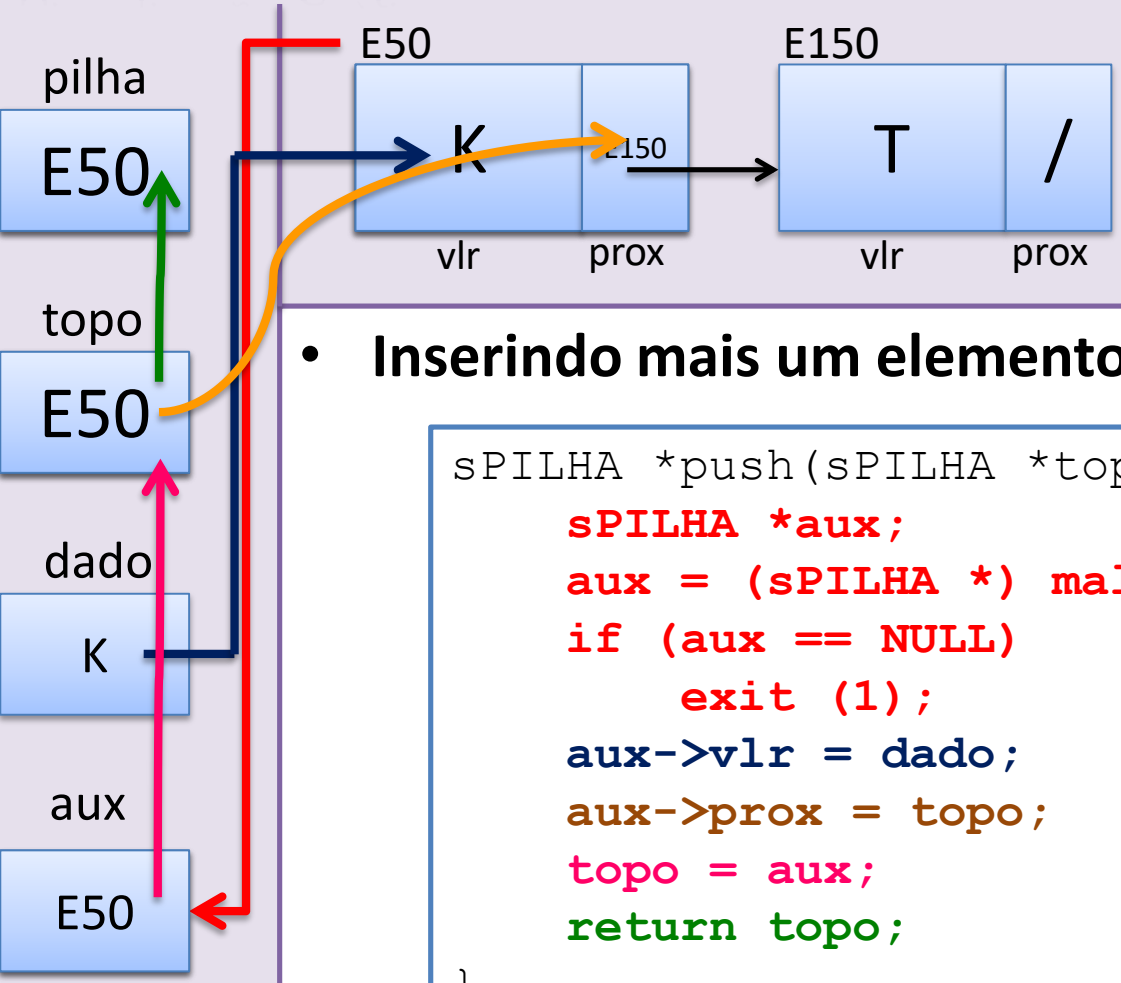
Acompanhe agora os passos para inserir um elemento no topo da pilha

- **Passo 5**: O endereço da lista com o novo dado é retornado para a variável "pilha" e as variáveis locais são eliminadas

```
sPILHA *push(sPILHA *topo, char dado){
     sPILHA *aux;
     aux = (sPILHA *) malloc (sizeof(sPILHA));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
     aux->prox = topo;
     topo = aux;
     return topo;
}
```

Implementação – Pilha – Inserir (push)

Visualização Gráfica



- **Inserindo mais um elemento no topo da pilha**

```
sPILHA *push(sPILHA *topo, char dado){
     sPILHA *aux;
     aux = (sPILHA *) malloc (sizeof(sPILHA));
     if (aux == NULL)
          exit (1);
     aux->vlr = dado;
     aux->prox = topo;
     topo = aux;
     return topo;
}
```

Implementação – Pilha – Mostra topo

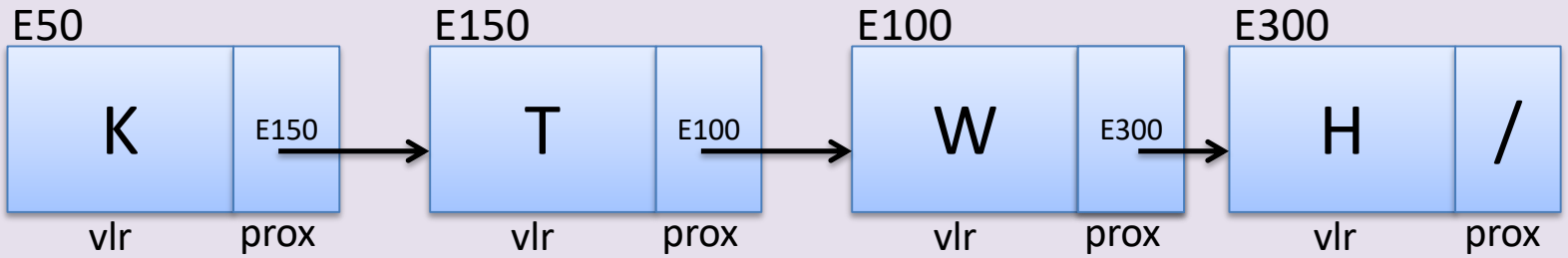
Visualização Gráfica

pilha

E50

topo

E50



- **Mostrando os elementos da lista**

```
void mostrar_topo(sPILHA *topo){
     if (estaVazia(topo))
        printf("\nPilha Vazia!\n\n");
     else
        printf("TOPO = %c\n\n", topo->vlr);
}
```

Implementação – Pilha – Excluir (pop)

Visualização Gráfica

pilha

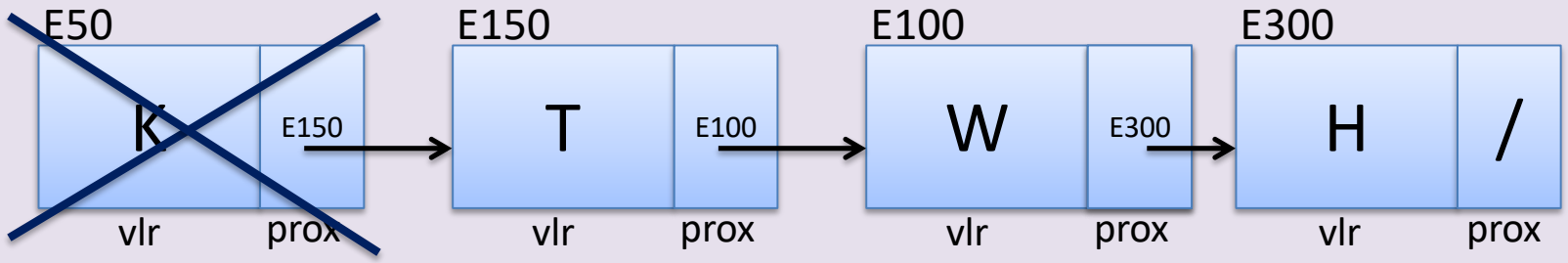
E150

topo

E150

aux

E50



- **Excluindo o elemento do topo da pilha**

```
sPILHA* pop(sPILHA *topo){
      sPILHA *aux;
      if (estaVazia(topo))
         printf("\n\nPilha VAZIA!\n\n");
      else{
         aux = topo;
         topo = topo->prox;
         free(aux);
      }
      return topo;
}
```

Aplicações

- **Exemplos de aplicações:**
  - **Pilha de execução de programas**
  - **Avaliação de expressões**
    - **verifique nos slides a seguir**

Infixa, Prefixa e Posfixa

- Infixa = A + B
- Prefixa ou notação polonesa = +AB
- Posfixa ou notação polonesa reversa = AB+

Posfixa

- Outros Exemplos:

a) A + B * C =

b) A * (B + C) / D – E =

c) A+B*(C-D*(E-F)-G*H)-I*3 =

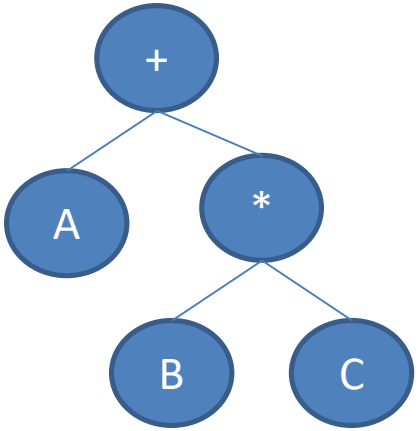
d) (A + B) * D + E / (F + A * D) + C =

e) A/B*C+D*E-A*C =

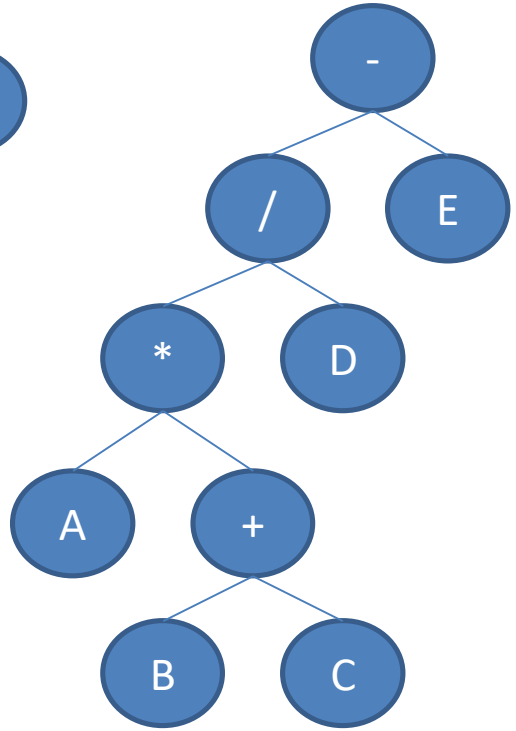
f) A*(B+C*A-D)*D =

- Outros Exemplos:

a) A + B * C = ABC*+



b) A * (B + C) / D – E = ABC+*D/E-

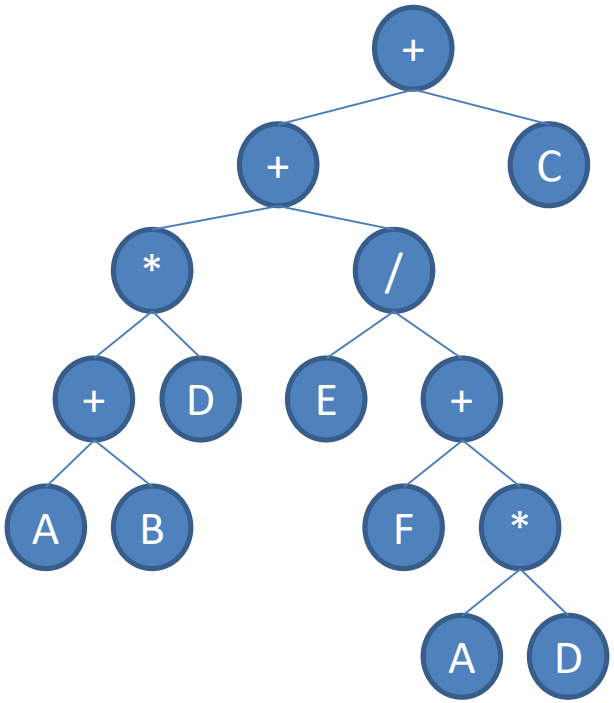


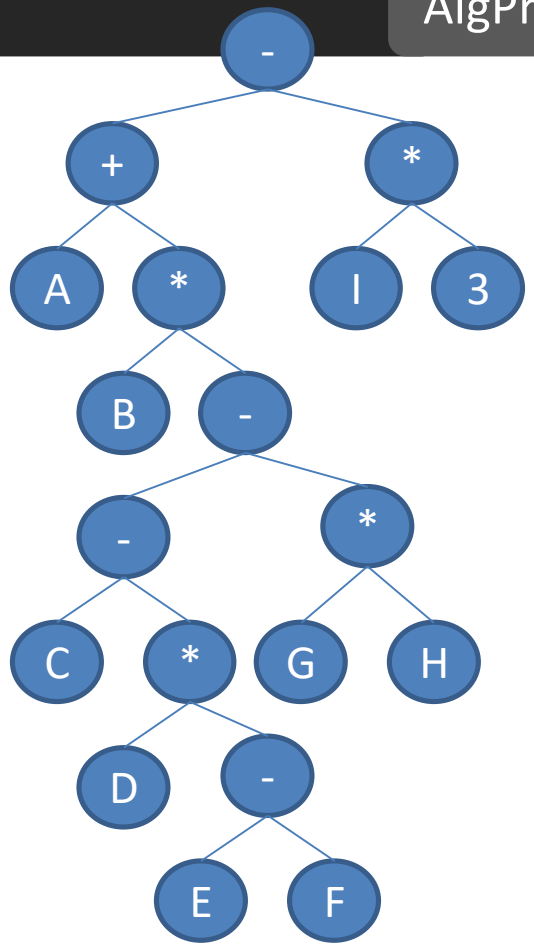
Posfixa

- Outros Exemplos:

  c) A+B*(C-D*(E-F)-G*H)-I*3 =

  ABCDEF-*-GH*-*+I3*-



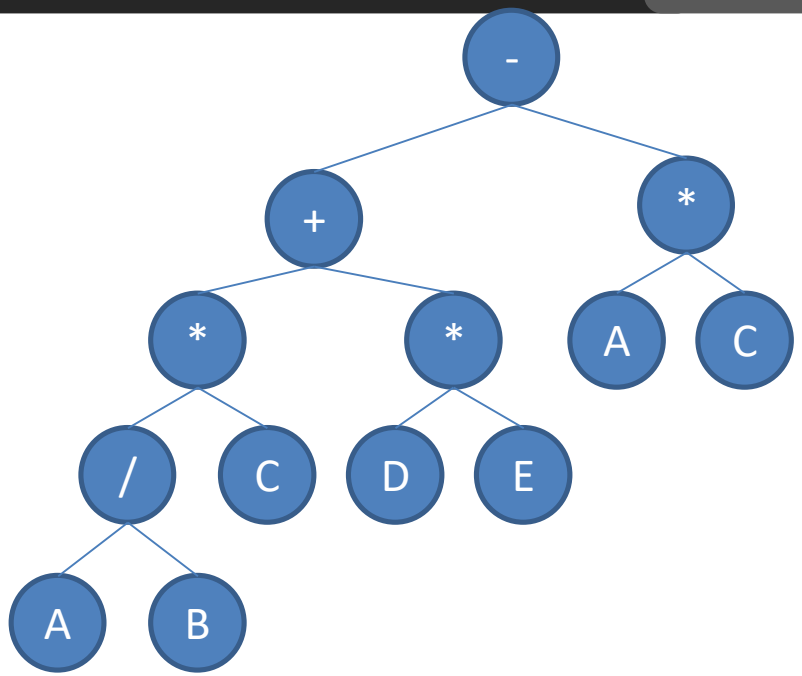


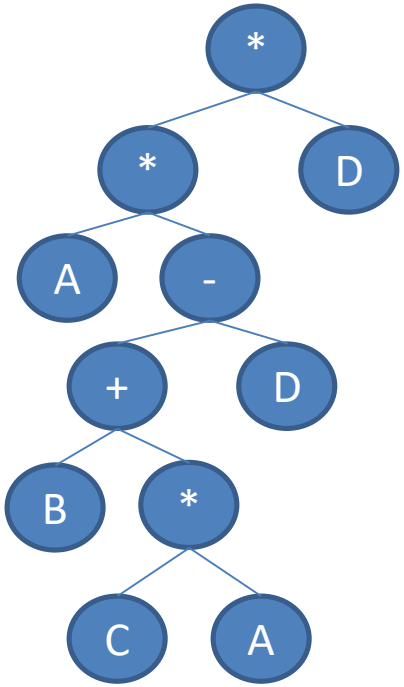
c) (A+B) * D + E / (F+A*D) + C =

AB+D*EFAD*+/+C+

- Outros Exemplos:

e) A/B*C+D*E-A*C =
AB/C*DE*+AC*-





f) A*(B+C*A-D)*D =
ABCA*+D-*D*

Problema a ser resolvido

- Desenvolver um programa que leia uma expressão infixa, transforme-a para posfixa e calcule o resultado. Utilizando pilha dinâmica. Só serão aceitos:
  - números de 1 a 9
  - operadores: +, -, * e /
  - “(“ e “)”
  - Letras representando as variáveis da expressão: “A”..”Z” (Obs.: se o usuário digitar minúsculo, transforme em maiúsculo.

Programa - Etapas

- MENU
  - Inserir expressão
  - Calcular expressão
  - Sair do programa

- Inserir Expressão:
  - Inicialização:
    - pilha de operadores: onde serão armazenados os operadores (+, -, /, *) na conversão para expressão posfixa (notação polonesa reversa)

- Inserir Expressão:
  - Ler a expressão no modelo “Infixa” (A+B)
  - Avaliar a expressão, caso não esteja correta, ler outra.
    - Regras para a avaliação da expressão “Infixa”:

      1) O 1º símbolo não pode ser um operador (+, -, *, /) - Ex.: -A*B

- Inserir Expressão:
  - Regras para a avaliação da expressão "Infixa":

    2) Um operando (A..Z) ou (1..9) não pode ser precedido por outro ou por parênteses de fechamento

    - Ex.1: AB-C
    - Ex.2: A+(B-C)D
    - Ex.3: 3+3B

- Inserir Expressão:
  - Regras para a avaliação da expressão "Infixa":

    3) Um operador (+, -, *, /) não pode ser precedido por outro, nem por parêntese de abertura "("

    - Ex.1: A*/B

    - Ex.2: A+(-B-C)

- Inserir Expressão:
  - Regras para a avaliação da expressão "Infixa":

    4) Um parêntese de abertura "(" não pode ser precedido por um de fechamento ")", nem por operando (A..Z)

    - Ex.1: (A+B)(C-D) ¤ Ex.2: A(B-C)

- Inserir Expressão:
  - Regras para a avaliação da expressão "Infixa":

    5) Um parêntese de fechamento ")" não pode ser precedido por um de abertura "(", nem por operador (+, -, *, /)

    - Ex.1: A()+B

    - Ex.2: (A+)-B

- Inserir Expressão:
  - Regras para a avaliação da expressão “Infixa”:

    6) A quantidade de parênteses de abertura “(“ deverá ser igual a quantidade de parênteses de fechamento “)”

    - Ex.1: (A+B)*C)

- Inserir Expressão:
  - Regras para a avaliação da expressão “Infixa”:

    7) O último símbolo não poderá ser um operador

    (+, -, *, /)

    - Ex.1: A/B+

    8) Não poderá existir qualquer caracter diferente de “(“, “)”, “A”..”Z”, “1”..”9”, “+”, “-”, “/”, “*”

- Inserir Expressão:
  - Transformar a expressão “infixa” (A+B) para a notação polonesa reversa “Posfixa” (AB+)
    - Regras para passar a expressão de “Infixa” para NPR -“Posfixa”:

      1) 0..9 ➔ coloca para a expressão NPR

- Inserir Expressão:
  - Regras para passar a expressão de “Infixa” para NPR -“Posfixa”:

    2) A..Z➔ coloca para a expressão NPR

    3) “(“ ➔ empilha

Programa - Etapas

- Inserir Expressão:
  - Regras para passar a expressão de “Infixa” para NPR -“Posfixa”:

    4) +, -, *, / ➔ a) pilha vazia: empilha; b) se o que estiver no topo tiver prioridade maior ou igual, desempilha e coloca na expressão NPR e aí empilha o operador

    -> (➔ Prioridade = 1

    -> +, - ➔ Prioridade = 2

    -> *, / ➔ Prioridade = 3

- Inserir Expressão:
  - Regras para passar a expressão de “Infixa” para NPR -“Posfixa”:

    5) “)” ➔ desempilha cada operador, colocando-os na expressão NPR, até encontrar um parêntese de abertura “(“, sendo este também desempilhado, porém não incluído na NPR

- Inserir Expressão:
  - Regras para passar a expressão de “Infixa” para NPR

    -“Posfixa”:

    6) Ao terminar a expressão, desempilha tudo que há ainda na pilha e coloca na expressão NPR

- Calcular Expressão:

  1) Inicializar:

     - pilha dinâmica de valores: onde serão armazenados os valores dos operandos (A..Z) e os resultados dos cálculos
     - LDSE de operandos (A..Z)

Programa - Etapas

- Calcular Expressão:

  2) Ler os valores dos operandos e armazená-los na LDSE

  3) Gerar o resultado da expressão, lendo a NPR do início:

     a) se for um número de 0..9, transforme-o para dado numérico e empilha

     b) se for um operando A..Z, empilha seu respectivo valor retirado

Programa - Etapas

- Calcular Expressão:

  3) Gerar o resultado da expressão, lendo a NPR do início:

     c) se for um operador (+, -, *, /), desempilha dois valores, execute o respectivo calculo e empilha o resultado

  4) No final, o único valor que sobrou na pilha será o resultado da expressão

Leitura Recomendada

- DROZDEK, Adam. Estrutura de Dados e Algoritmos em C++. Editora Pioneira Thomson Learning, 2005.
  - Pág 123 (Pilha)

- TENENBAUM A., LANGSAM Y. e AUGENSTEIN M. J. Estrutura de Dados usando C. Editora Makron, 1995.
  - Pág 86 (Pilha)

- FEOFILOFF, Paulo. Algoritmos em Linguagem C. Editora Campus, 2009.
  - Pág 39 (Pilha)

Exercício

- Desenvolva a função de transformar de Infixa

  para Posfixa