SCC 202 - Algoritmos e Estruturas de Dados I

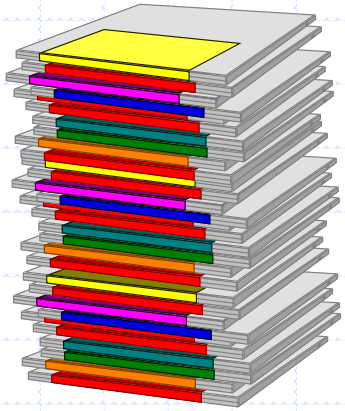
TAD Pilha

Lembrem...TADs são tipos definidos em termos de seu comportamento e não de sua representação (que pode variar na busca de eficiência)

12/8/2010

Pilha

- O que é?
- Para que serve?



Exemplo da vida real de estudante...

- Ex: Pilhas de bandeja do Bandejão

  Bandejas inicialmente são empilhadas

  Pega-se (remove-se) bandeja do topo

  Se não há mais bandejas, a pilha está vazia e não podemos remover mais

  Desde que nós podemos ver a bandeja do topo, se ela estiver suja podemos não querer pegar.

- Este ex. faz referência a 4 operações de pilhas:

  Inserir = push

  Remover = pop

  Se a pilha não contém elementos, a verificação de vazia retorna true, caso contrário false

  Topo_da_pilha = top_of_stack, retorna o topo da pilha (cópia) para ser examinado, sem removê-lo.

Pilha (*stack*)

- Definição
  - *Estrutura para armazenar um conjunto de elementos que funciona da seguinte forma*
    - Novos elementos sempre entram no "topo" da pilha
    - O único elemento que se pode retirar da pilha em um dado momento é o elemento do topo
- Para que serve
  - Para modelar situações em que é preciso "guardar para mais tarde" vários elementos e "lembrar" sempre do último elemento armazenado
- L.I.F.O.
  - *Last In, First Out*

Problema: chamada de sub-rotinas

Rotina A
1 print “A”
2 call C
3 call B
4 call D
5 return

Rotina B
1 call C
2 print “B”
3 call D
4 call C
5 return

Rotina C
1 print “C”
2 call D
3 return

Rotina D
1 print “D”
2 return

Qual o resultado da execução da rotina A?

Problema: chamada de sub-rotinas

Rotina A
1 print “A”
2 call C
3 call B
4 call D
5 return

Rotina B
1 call C
2 print “B”
3 call D
4 call C
5 return

Rotina C
1 print “C”
2 call D
3 return

Rotina D
1 print “D”
2 return

Qual o resultado da execução da rotina A?

Qual a dificuldade para se fazer esse cálculo?

Problema: chamada de sub-rotinas

Rotina A
1 print “A”
2 call C
3 call B
4 call D
5 return

Rotina B
1 call C
2 print “B”
3 call D
4 call C
5 return

Rotina C
1 print “C”
2 call D
3 return

Rotina D
1 print “D”
2 return

Qual o resultado da execução da rotina A?

Qual a dificuldade para se fazer esse cálculo?

Possíveis soluções?

Problema: chamada de sub-rotinas

1. Um computador está executando a rotina X e, durante a execução de X, encontra uma chamada à rotina Y

2. Para-se a execução de X e se inicia a execução de Y

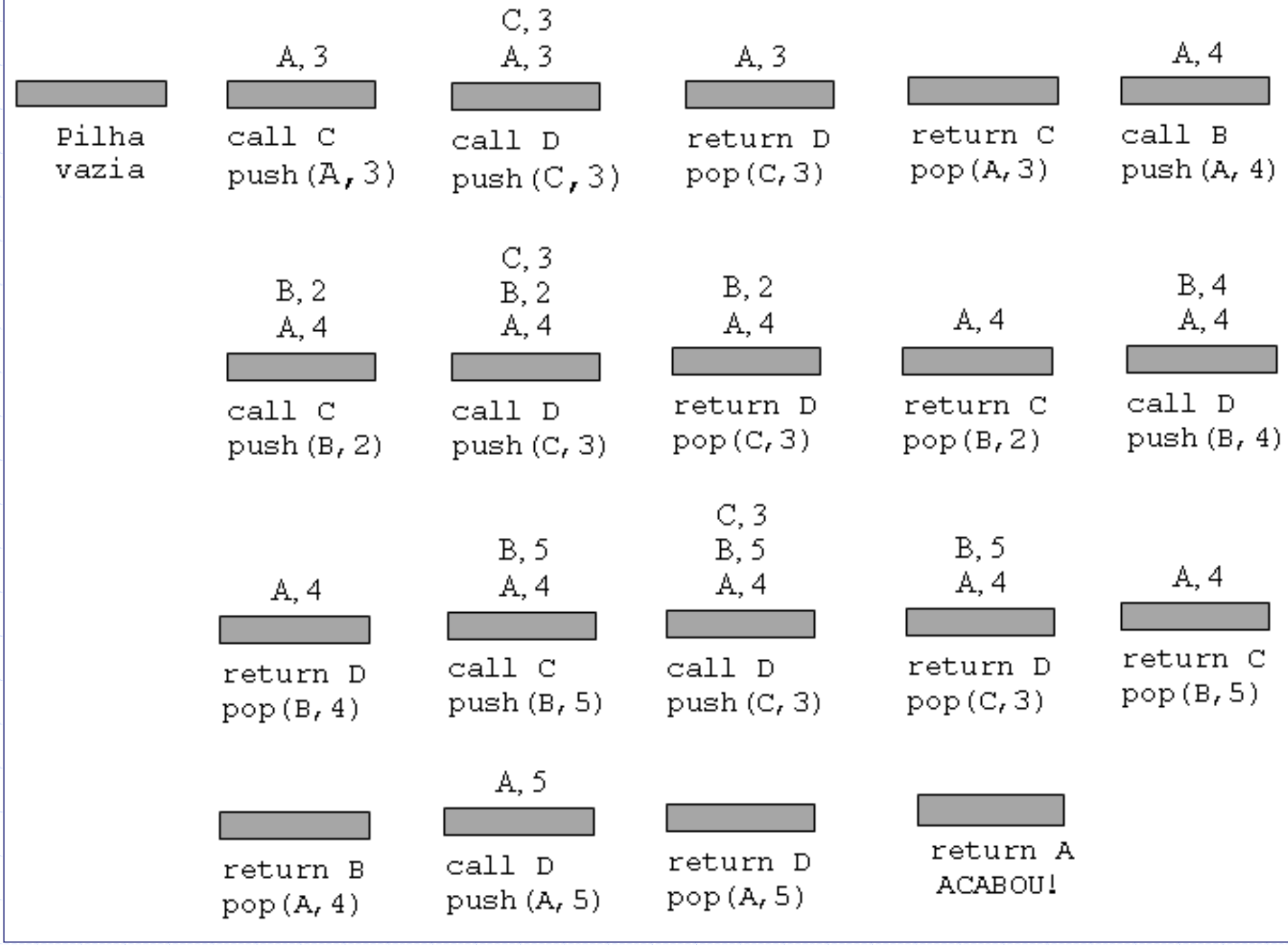
3. Quando se termina a execução de Y, o computador deve saber o que fazer, isto é, onde voltar na rotina X

Problema: chamada de sub-rotinas

- Dificuldade
  - O que estava sendo executado quando uma sub-rotina foi interrompida? Para onde voltar agora que se chegou ao fim de uma sub-rotina?
- Solução
  - A cada chamada de sub-rotina, armazenar o endereço de retorno (rotina e número da linha, por exemplo)
  - Como armazenar o endereço de retorno de chamadas sucessivas: pilha

Problema: chamada de sub-rotinas

- A cada comando call
  - Empilha (*push*) o endereço para retornar depois
  - Passa a executar a nova sub-rotina
- A cada comando return
  - Desempilha (pop) o último endereço armazenado
  - Passa a executar a partir do endereço desempilhado


Pilha vazia
A, 3
call C
push(A, 3)
C, 3
A, 3
call D
push(C, 3)
A, 3
return D
pop(C, 3)
return C
pop(A, 3)
A, 4
call B
push(A, 4)
B, 2
A, 4
call C
push(B, 2)
C, 3
B, 2
A, 4
call D
push(C, 3)
B, 2
A, 4
return D
pop(C, 3)
A, 4
return C
pop(B, 2)
B, 4
A, 4
call D
push(B, 4)
A, 4
return D
pop(B, 4)
B, 5
A, 4
call C
push(B, 5)
C, 3
B, 5
A, 4
call D
push(C, 3)
B, 5
A, 4
return D
pop(C, 3)
A, 4
return C
pop(B, 5)
return B
pop(A, 4)
A, 5
call D
push(A, 5)
return D
pop(A, 5)
return A
ACABOU!

Problema: chamada de sub-rotinas

- Resultado
  - A, C, D, C, D, B, D, C, D, D

Pilha

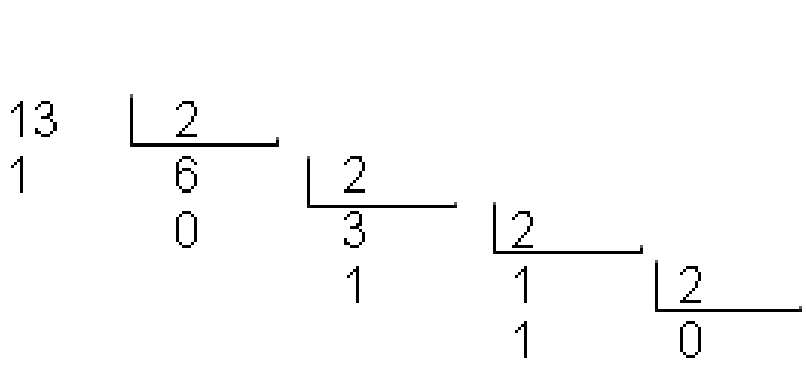
- Operações usuais
  - Push(P,X): empilha o valor da variável X na pilha P
  - Pop(P,X): desempilha P e retorna em X o valor do elemento que estava no topo de P
  - X=top(P): acessa o valor do elemento do topo de P, sem desempilhar
  - P = Create(): cria uma pilha vazia P
  - Empty(P): esvazia uma pilha P

Outras rotinas?

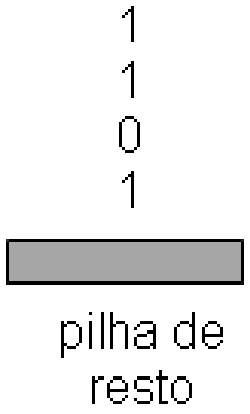
- Y = Size(P): Y recebe o número de elementos de P
- Y = IsFull(P): Y recebe *true* se a pilha estiver cheia; *false* caso contrário
- Y=IsEmpty(P): Y recebe true se a pilha estiver vazia; false caso contrário

Exercício

- Fazer algoritmo de conversão decimal para binário usando pilha

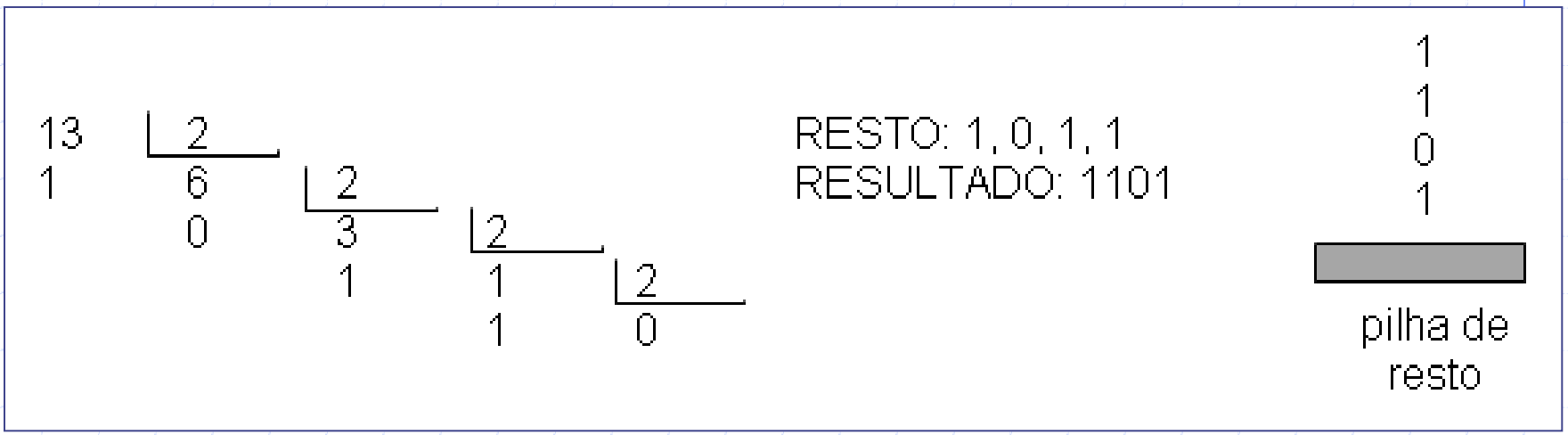


RESTO: 1, 0, 1, 1
RESULTADO: 1101



Exercício

- Fazer algoritmo de conversão decimal para binário usando pilha



Estratégia de resolução:

- a cada divisão, empilha o resto
- quando acabar a divisão (quociente=0), desempilha e escreve todos os elementos

Considerar prontos: operações da pilha, resto(X,Y) e quociente(X,Y)

Exercício

```
Variáveis
    P: pilha
    N: inteiro {número a ser convertido}
    X: inteiro {resto da divisão}

Início do algoritmo
    leia N
    create(P)
    repita
            X=resto(N,2)
            push(P,X)
            N=quociente(N,2)
    até que (N=0)
    escreva "o resultado é "
    enquanto (IsEmpty(P)=falso) faça
            pop(P,X)
            escreva X
Fim
```

Representação da pilha

- Representação: seqüencial ou encadeada
  - Seqüencial: Os elementos da pilha ficam, necessariamente, em seqüência (um ao lado do outro) na memória – posições contíguas.
  - Encadeada: Os elementos são encadeados com o seguinte por meio de um apontador, permitindo usar posições não contíguas de memória.

Implementação

- Representação: seqüencial ou encadeada
  - Seqüencial: Os elementos da pilha ficam, necessariamente, em seqüência (um ao lado do outro) na memória – posições contíguas.
  - **Implementação estática**
    - Todo o espaço de memória a ser utilizado pela pilha é reservado (alocado) em tempo de compilação
    - Desvantagem: Todo o espaço reservado permanece reservado durante todo o tempo de execução do programa, independentemente de estar sendo efetivamente usado ou não; pode não ser suficiente...
    - Inserção após o último com custo constante.
    - Remover o último feita com custo constante.
  - **Implementação dinâmica**
    - Vantagem de poder alocar o tamanho em tempo de execução e escapar do limite de um vetor de tamanho N.
    - Nesta solução, aloca-se novo bloco de tamanho N toda vez que o espaço do bloco anterior termina (esta checagem seria feita em PUSH). Aqui, o tamanho do vetor não é mais constante e ainda temos o benefício do acesso indexado.

Exemplo de declaração de vetor em C

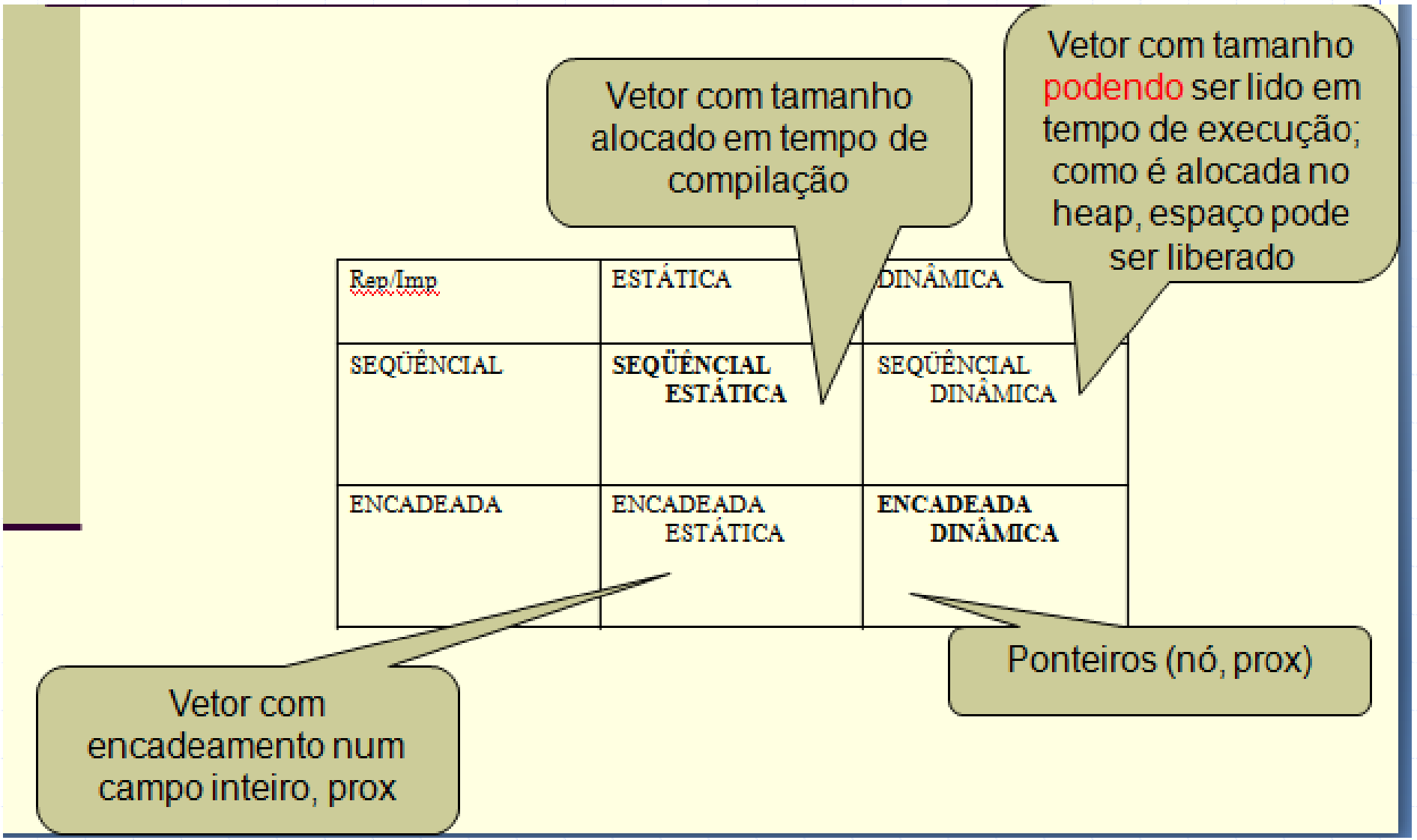
- alocação dinâmica de um vetor v de inteiros com 10 elementos em C
  - v armazena endereço inicial de uma área contínua de memória, suficiente para armazenar 10 valores inteiros
  - v pode ser tratado como um vetor declarado estaticamente
  - v aponta para o inicio da área alocada
  - v[0] acessa o espaço para o primeiro elemento
  - v[1] acessa o segundo
  - .... até v[9]

Implementação

- Representação: seqüencial ou encadeada

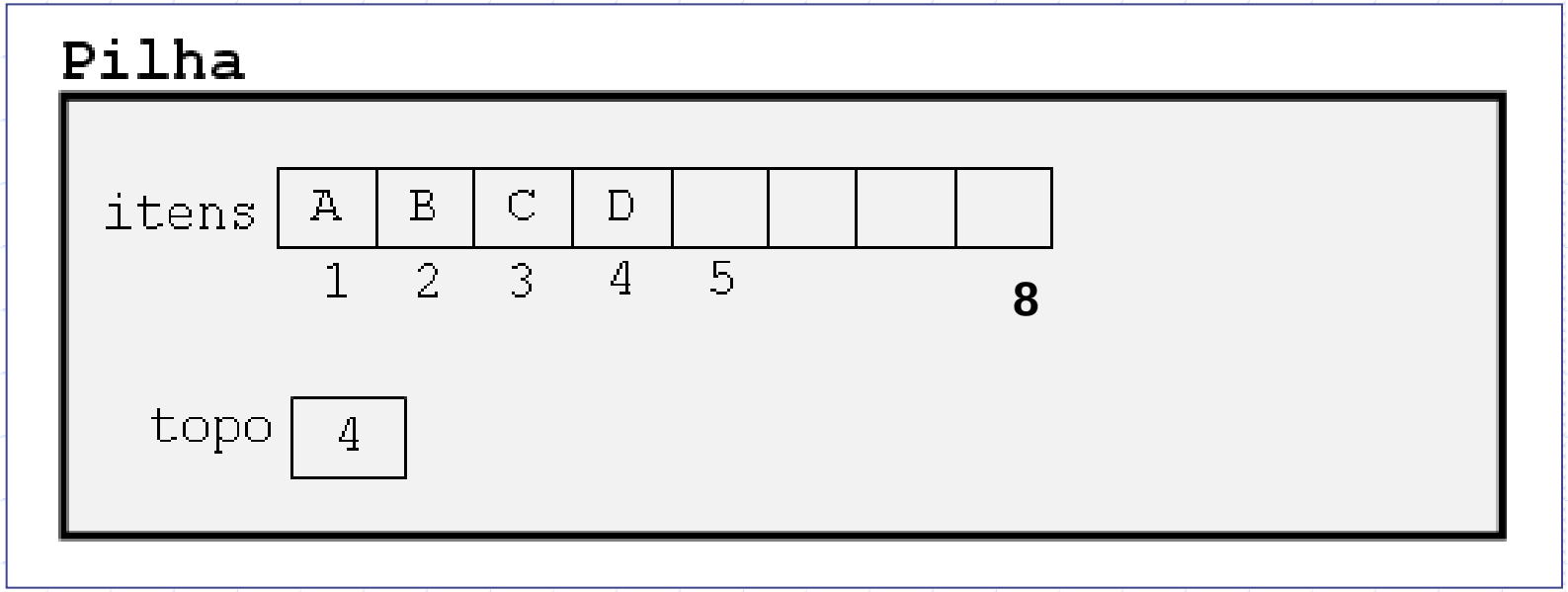
  - Encadeada: Os elementos são encadeados com o seguinte por meio de um apontador, permitindo usar posições não contíguas de memória.
    - Útil quando não existe previsão sobre o crescimento da pilha (ponteiros)
    - Desvantagem: utilização de memória extra para guardar os ponteiros

  - **Implementação: estática (vetor) ou dinâmica (ponteiro)**


Vetor com tamanho alocado em tempo de compilação
Vetor com tamanho podendo ser lido em tempo de execução; como é alocada no heap, espaço pode ser liberado
Rep/Imp
ESTÁTICA
DINÂMICA
SEQÜÊNCIAL
SEQÜÊNCIAL ESTÁTICA
SEQÜÊNCIAL DINÂMICA
ENCADEADA
ENCADEADA ESTÁTICA
ENCADEADA DINÂMICA
Vetor com encadeamento num campo inteiro, prox
Ponteiros (nó, prox)

Imp/REP da pilha

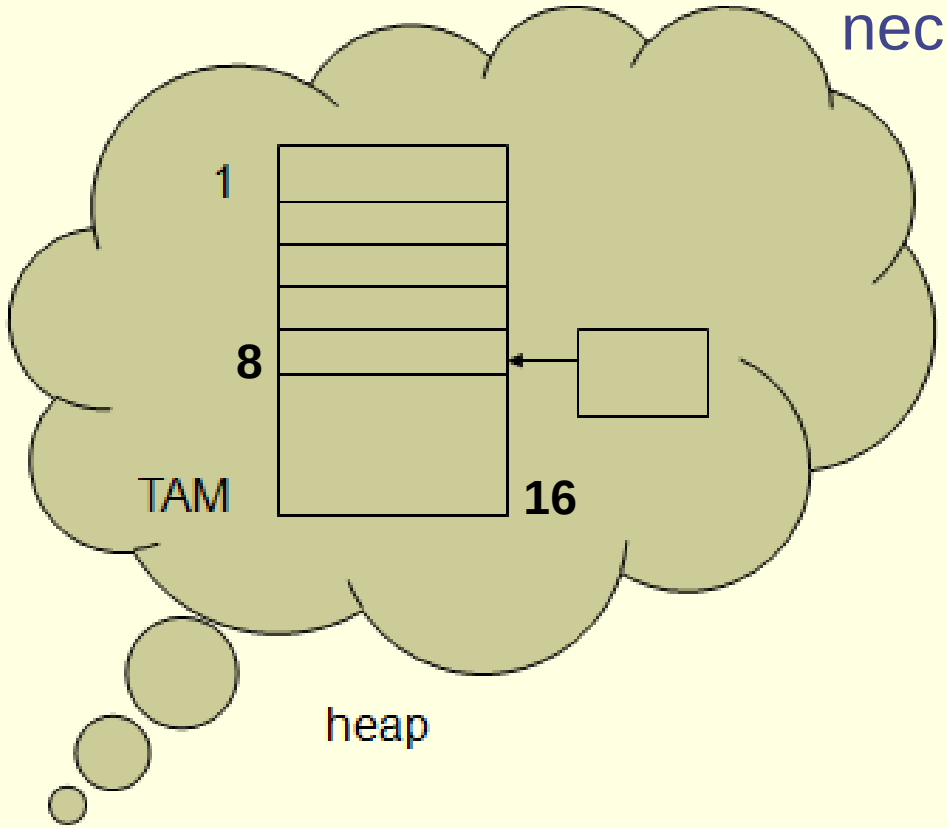
- Seqüencial e estática



Rep/Imp da pilha

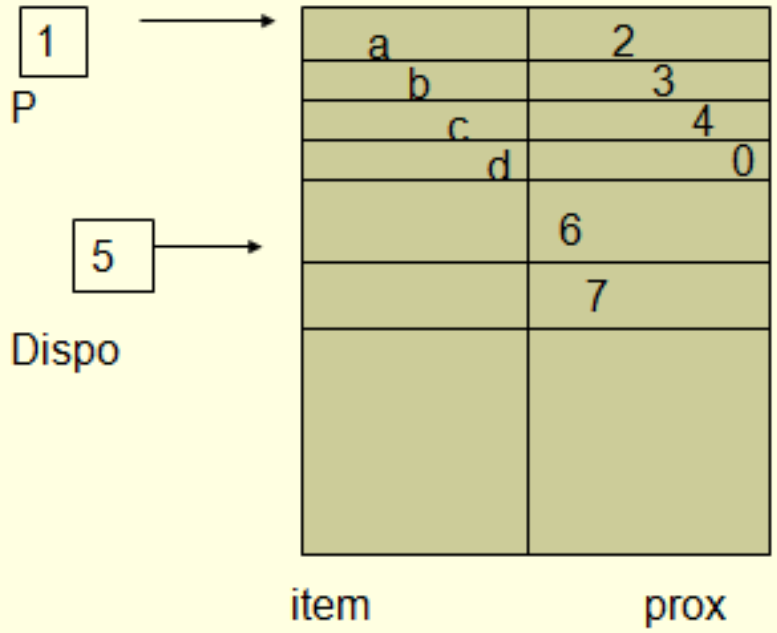
- Seqüencial e dinâmica

Aloca-se mais um bloco, quando necessário



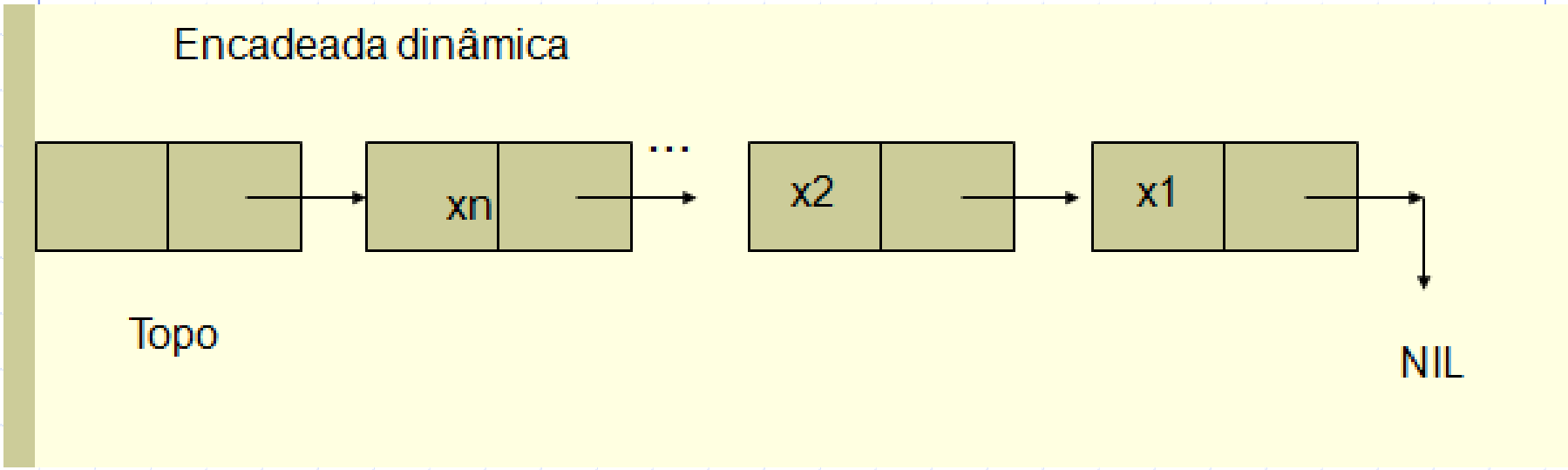
IMP/REP da pilha

- Encadeada Estática em Pascal




```
Space: array [1..TAM] of
        record
                item: tipo_item;
                prox: integer
        end;
P, Dispo: integer;
```

IMP/Rep da Pilha



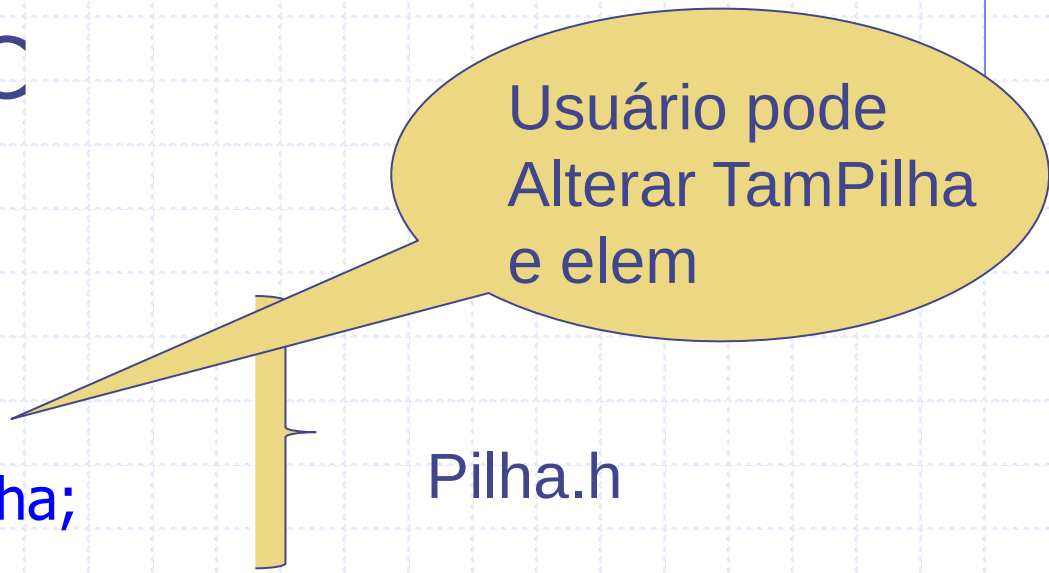
Exercício: Implementação da pilha sequencial e estática

- Declaração em C escondendo a ED do cliente ??

- Arrays são implementados internamente como ponteiros, em C.
  - Mas a idéia de chamar esta implementação de estática é que o tamanho da pilha é definido previamente e não pode ser alterado nesta implementação.

Implementação da pilha

- Declaração em C




```
#define TamPilha 100
typedef char elem;
typedef struct pilha Pilha;
```

Pilha.h

```
struct pilha{
        int topo;
        elem itens[TamPilha];
};
```

Pilha.c

Agradecimentos

- Thiago Pardo por parte do material