UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Bauru

Faculdade de Ciências

Disciplina: Estruturas de Dados I – **ED1**
Professora: Dra Simone das Graças Domingues Prado
e-mail: simone.prado@unesp.br

Apostila 01 – Pilha

Objetivos:

⇒ Conceituar tipo abstrato de dados: pilha
⇒ Conhecer e implementar vários tipos pilhas
⇒ Trabalhar com pilhas estáticas
⇒ Trabalhar com pilhas dinâmicas

Conteúdo:

1. Introdução
2. Pilhas estáticas
3. Pilhas dinâmicas
4. Exercícios

Departamento de Computação
Av. Engº Luiz Edmundo Carrijo Coube 14-01 Vargem Limpa Bauru-SP CEP: 17033-360
Fone: (14) 3103-6034 Fax: (14) 3103-6079 email: dcogeral@fc.unesp.br

1. Introdução

Vetores e vários tipos de listas encadeadas já foram estudados. Percebe-se que, nos vetores e nas listas encadeadas, o acesso era feito sem nenhuma restrição. Podia-se inserir e remover elementos em qualquer posição, só dependendo se os dados eram ordenados ou não.

As Pilhas e as Filas são conjuntos de elementos (que podem ser representados por vetores ou listas encadeadas) que possuem critérios de acessos restritivos. Vejamos os critérios de acesso mais usuais:
FIFO - (First Input, First Output) ou seja, o primeiro a entrar, é o primeiro a sair.
LIFO - (Last Input, First Output) ou seja, o último a entrar, é o primeiro a sair.

Levando em consideração esses critérios de acesso e pelo que se conhece, informalmente, por pilha e fila pode-se identificar que uma Pilha é uma lista que tem como critério de acesso o LIFO, e a fila, o FIFO.

Dessa forma, uma Pilha é um conjunto de elementos no qual novos itens podem ser inseridos ou removidos em uma única extremidade, a qual é chamada de topo. A pilha possuirá características bem próprias de acesso e por consequência terá operações bem simples de empilhar e desempilhar elementos.

Veja a figura 01, onde se têm quatro peças empilhadas rotuladas como A, B, C e D. Para a pilha abaixo ficar com essa configuração deve-se colocar primeiro a peça A, depois a B, C e D. Ao retirar só conseguiremos a ordem: D, C, B e A

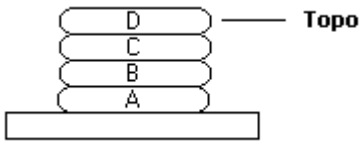


Figura 01. Uma pilha com elementos A, B, C e D

A manipulação de uma pilha pode ser feita de forma estática ou dinâmica. A forma estática usará variável do tipo array em C e as pilhas dinâmicas usarão ponteiros. Numa pilha estática tem-se a preocupação com o tamanho da pilha.. Assim, ao inserir um elemento deve-se verificar se a pilha comporta. Na pilha dinâmica (que usa ponteiros) ao inserir o elemento tem-se de verificar se existe memória para a criação de um novo elemento na pilha.

2. Pilhas estáticas

Na implementação de pilhas estáticas pode surgir várias formas de construção do tipo de dado pilha. Veja aqui duas definições: uma definição usará um vetor e variável topo separados e a outra definição usará uma estrutura que conterá o vetor e o topo.

Independente de qual seja a construção do tipo pilha, deve-se preocupar com as operações básicas de:

- verificação de pilha vazia (se estiver vazia, não se pode tentar desempilhar elementos)
- verificação de pilha cheia (se estiver cheia, não se pode empilhar elementos)
- empilhar um elemento
- desempilhar um elemento e devolvê-lo
- visualizar a pilha

2.1. Primeira definição de Pilha Estática

Trabalha-se com a ideia de ter um vetor de inteiros e o topo é controlado por uma variável inteira fora desse vetor.

```
#define MAX 100
typedef int def_pilha[MAX];
```

A referência é feita da seguinte forma para:

```
def_pilha pilha;
int topo = -1; //valor inicial – pilha vazia
```

Topo => topo
Elemento do topo => pilha[topo]

As rotinas estão disponíveis em PilhaEst01.c

2.2. Segunda definição de Pilha estática

Essa definição trabalha com uma estrutura que contem um vetor de elementos e uma variável inteira para guardar a posição do topo. Todas as operações são executadas em cima dessa estrutura.

```
#define MAX 100
typedef struct {
        int  topo;
        int  elementos[MAX];
        } def_pilha;
```

A referência é feita da seguinte forma para:

```
def_pilha pilha;
```

Topo => pilha.topo
Elemento do topo => pilha.elementos[pilha.topo]

As rotinas estão disponíveis em PilhaEst02.c

3. Pilhas dinâmicas

Na implementação de pilhas dinâmicas pode surgir várias formas de construção do tipo de dado pilha. Aqui veremos duas definições: uma definição usará um **Lista Linear Simplesmente Encadeada** e outra, uma **Lista Linear Simplesmente Encadeada com Descritor**.

3.1. Pilhas usando Lista Linear Simplesmente Encadeada

Definição:

```
typedef struct no_pilha{
        int  info;
        struct no_pilha *prox;
        } *def_pilha;
```

Aqui a inserção e remoção serão feitas no **início** da lista encadeada para a implementação da pilha.

```
                    def_pilha Pilha;
Elemento do topo =>  Pilha  ->  info;
```

As Rotinas estão disponíveis em PilhaLinear.c

3.2. Pilhas usando Lista Linear Simplesmente Encadeada com Descritor

```
typedef struct no_pilha{
    int  info;
    struct no_pilha *prox;
    } No;
typedef struct descritor{
    int quantidade;
    No *pilha;
    } *def_pilha;
```

Da mesma forma, a inserção e remoção serão feitas no **início** da lista.

def_pilha *Pilha
Elemento do topo => Pilha -> pilha -> info;

As Rotinas estão disponíveis em PilhaLinearDescr.c

4. Exercícios:

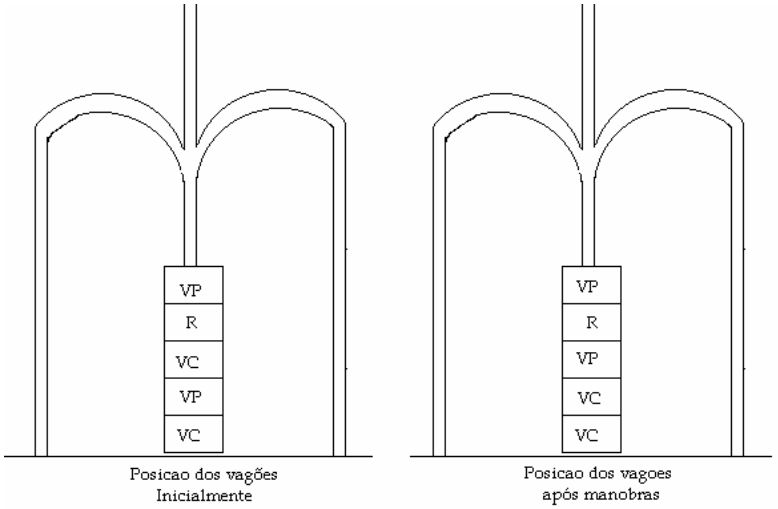
1) Construir uma pilha que usa somente um vetor (de tamanho MAX), onde pilha[0] contém o controle do topo da pilha e pilha[1] a pilha[MAX-1] contêm os elementos da pilha. Defina as rotinas de verificação de pilha vazia, verificação de pilha cheia, empilhar um elemento, desempilhar um elemento e visualizar a pilha

2) Em algumas aplicações tem-se que trabalhar com mais de uma pilha ao mesmo tempo. Podem-se implementar essas pilhas num mesmo vetor, de forma que cada uma ocupe parte desse vetor. Faça um programa que tenha duas pilhas alocadas em um mesmo vetor, onde cada uma cresce em sentido oposto, sem que as duas fiquem totalmente ocupadas simultaneamente. Implemente as operações básicas de verificação se uma pilha está vazia, verificação se uma pilha está cheia, empilhar um elemento em cada uma das pilhas, desempilhar um elemento de cada uma das pilhas e visualizar as pilhas.

3) Um **estacionamento é composto de um único beco** que guarda no máximo N carros. Existe apenas uma entrada/saída no estacionamento, em uma extremidade do beco. 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, e os outros carros voltarão a ocupar a sequencia inicial. Escreva um programa que processe um grupo de cartões do estacionamento.
   O programa deve imprimir uma mensagem sempre que um carro chegar ou sair. Quando um carro chegar, a mensagem deve especificar se existe ou não vaga para o carro no estacionamento. Se não houver vaga, o carro partirá sem entrar no estacionamento.
   Quando um carro sair do estacionamento, a mensagem deverá incluir o número de vezes em que o carro foi manobrado para fora e para dentro do estacionamento para permitir que outros carros saíssem.

4) Siga as instruções abaixo:
   (a) Crie o tipo da pilha.
   (b) Crie a rotina de inserção de um elemento na pilha.
   (c) Crie a rotina de remoção de um elemento na pilha. Esta rotina deve devolver 1-True/0-False, o valor do elemento retirado da pilha e a pilha modificada.
   (d) Crie a rotina de visualização da pilha
   (e) Simule manobras numa estação ferroviária de forma a **montar uma locomotiva** para sair de viagem. Imagine que inicialmente são colocados num terminal vários vagões misturados (vagões de carga - VC, vagões de transporte - VT e vagão restaurante R). Ao montar a locomotiva, os vagões são manipulados em duas outras linhas, além da linha central, para poder separar os vagões em três partes diferentes e depois recolocá-los em ordem. Na montagem da locomotiva se coloca um vagão de passageiros na frente, um vagão restaurante, os outros vagões de passageiros e depois os vagões de carga. Veja a figura ilustrativa abaixo. Faça uma rotina que faça essas manobras de forma que tendo uma pilha inicial de vagões ela seja transformada numa locomotiva. Use as rotinas empilha, desempilha. Não use uma variável auxiliar para percorrer a pilha. Imagine o caso real.
   (f) Crie o programa principal (main()) para testar as suas rotinas

   Podem ser deixados vagões restaurantes nas linhas, se houver mais de um para ser alocado.



5) Uma pilha pode ser usada para **rastrear os tipos de escopos** encontrados numa expressão matemática e verificar se o uso deles está correto. Os delimitadores de escopos podem ser os parênteses, os colchetes e as chaves. Escreva um programa que leia uma expressão matemática e verifique se os escopos estão posicionados de forma correta

6) Considere uma **gramática** que só aceite cadeias de caracteres da forma: xCy, onde x e y são cadeias de caracteres que possuem os símbolos A e B. Só que a cadeia y tem de ser a inversa de x. Então se x = AB, y será BA para que a cadeia ABCBA seja aceita por essa gramática.
Considere um programa que faça a leitura de uma cadeia qualquer e verifique se ela pertence a essa gramática. Para isso, o programa lê uma cadeia e a passa (por parâmetro) para a rotina VERIFICA() que retorna 1, se a cadeia pertence à gramática e 0 se não pertence.

7) Suponha uma calculadora que trabalha apenas com números não negativos, e que tem somente quatro operações: soma, subtração, produto e divisão inteira. A máquina tem 16 teclas, representadas pelos caracteres:
0 1 2 3 4 5 6 7 8 9 + - * / C E
onde C representa o "clear" e o E indica que vai ser fornecido um número. A máquina usa a notação polonesa sufixa, ou seja, o operador vem depois dos operandos. Escreva uma pilha de inteiros para simular essa calculadora: cada caracter que entra é tratado, e a resposta é o conteúdo da pilha da máquina. Inicialmente a pilha da máquina está vazia.
As ações correspondentes a cada caracter são:

i = 0 ... 9	: troque o valor de x do topo da pilha por x*10+i
E	: empilhe 0
Op = + - * /	: tire dois elementos Y e x da pilha e empilhe x op y
C	: esvazia a pilha

Exemplo-1: E 9 0 E 2 0 +
Então (90+20)

Exemplo-2: E 9 0 E 2 0 E 1 5 E 1 3 - * E 5 + /
Então: (90/((20*(15-13))+5))

Departamento de Computação
Av. Engº Luiz Edmundo Carrijo Coube 14-01 Vargem Limpa Bauru-SP CEP: 17033-360
Fone: (14) 3103-6034 Fax: (14) 3103-6079 email: dcogeral@fc.unesp.br

8) Suponha um armário com **gavetas de revistas**, onde cada gaveta abriga um único tipo de revista (mas pode guardar várias revistas de nomes diferentes). Assim, as gavetas são nomeadas pelos seus tipos: infantil, educação, feminina, atualidades, saúde e bem-estar, culinária, masculinas, negócios, moda, jovens, tecnologias e esporte.
Quando chega o carregamento, as revistas não estão em ordem e nem agrupadas. As informações da revista que são importantes são: tipo de revista, nome da revista e editora.

(a) Defina a estrutura que representará o armário onde serão guardadas as revistas
(b) Defina a estrutura que representará a gaveta. Dentro da gaveta é obedecido o conceito de pilha, já que uma revista é sobreposta a outra.
(c) Construa a rotina de Empilhamento de uma revista (tipo,nome,editora) em uma gaveta específica.
(d) Construa uma rotina para desempilhamento de uma revista de uma gaveta.
(e) Crie o programa principal (main()) para testar as suas rotinas

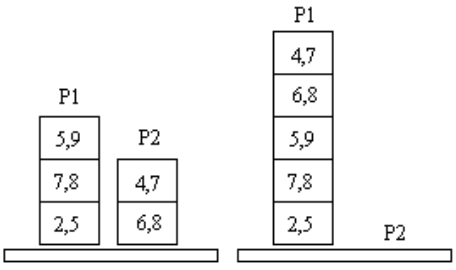
9) Imagine um **colecionador de vinhos** que compra vinhos recentes e guarda-os em uma adega para envelhecerem, e que a cada ocasião especial abre sempre sua última aquisição (para poupar os mais antigos). Construa um programa que:

a) Permita incluir novos vinhos na adega
b) Informe qual vinho deve ser aberto em uma ocasião especial
c) Relacione as cinco aquisições mais antigas

As informações básicas que o registro do vinho deve conter são: nome do produto e safra.

10) O problema das **Torres de Hanói** é bastante estudado em computação. O problema consiste em n discos de diferentes diâmetros e três estacas: A, B e C. Inicialmente os discos estão encaixados na estaca A onde o menor está sempre encima do maior disco. O objetivo é deslocar os discos para uma estaca C, usando a estaca B como auxiliar. Somente o primeiro disco de toda estaca pode ser deslocado. Isso dá a ideia de pilhas. Portanto construa a resolução desse exercício usando pilhas e para n = 4, ou seja, para 04 discos.

11) Considere a existência de um tipo PILHA de números de ponto flutuante. Implemente um programa que leia duas pilhas, P1 e P2, e passe todos os elementos da pilha P2 para o topo da pilha P1 sem trocar a ordem e sem ferir o conceito de pilha. A figura a seguir ilustra essa concatenação de pilhas:



12) Considere que a pilha use a estrutura de uma Lista Circular Simplesmente Encadeada. Implemente as operações básicas de: verificação de pilha vazia, verificação de pilha cheia, empilhar um elemento, desempilhar um elemento e visualizar a pilha

13) Existe um jogo de corrida que está sendo desenvolvido pela Empresa MTTK que promete balançar o mercado de jogos por computador. O jogo exige muita estratégia já que ao passar por algumas trilhas se você

estiver em primeiro poderá ser realocado para a outra posição. Uma das partes do jogo consiste em após uma trilha larga que pode ter até três carros emparelhados, ela torna-se uma trilha bem estreita forçando os motoristas a ficarem enfileirados. Essa trilha termina num rio que tem uma profundidade em que os carros não conseguem passar e chegar na trilha do outro lado. Não tem como eles ultrapassarem sem a ajuda de uma balsa. A balsa que trafega neste rio possui um único lado para entrada e saída dos carros. Ela comporta duas fileiras de 06 carros e assim, no máximo 12 carros. Durante o trajeto da travessia do rio a balsa vira e consegue desembarcar os carros do outro lado do rio. O desembarque exige a saída de um carro por vez, sendo que primeiro saem os carros da fileira da direita e depois os da esquerda. Ao desembarcar na nova trilha os motoristas percebem que ela tem duas pistas. Assim, na medida que eles vão saindo da balsa eles ficam na pista que tem menos carros.
Implemente essa parte do jogo seguindo as seguintes diretrizes:

(f) Defina a(s) estrutura(s) a ser(em) usada(s)
(g) Construa as rotinas (separadamente):
- entrada de um carro (por vez) na trilha 1 (quando saem da trilha larga e cai na trilha estreita)
- entrada de um carro na balsa (suponha que você tenha na corrida, no máximo, 12 carros competindo)
- saída de um carro da balsa (primeiro os carros da direita e depois os da esquerda da balsa)
- entrada de um carro na trilha 2 (quando saem da balsa encontram a trilha com 2 pistas)

(h) Construa a rotina que manipule as rotinas acima e realize essa parte do jogo.

14) Em um concurso, por falta de tempo hábil, os 30 (trinta) candidatos seriam enumerados de acordo com a ordem de chegada no dia da prova. A prova constava de duas fases: uma escrita e uma prática. A prova escrita seria feita numa sala (A) que possuía três fileiras de 10 carteiras. A prova prática seria no laboratório de informática (sala B) que possuía duas fileiras de 15 micros.
A dinâmica do processo foi o seguinte: assim que o candidato chegava, ele era encaminhado para a sala A. A entrada na sala era feita um por vez. Ao entrar na sala ele deveria sentar no lugar vago considerando a ocupação do fundo para frente e da esquerda para a direita. Ao terminar a prova o candidato deveria permanecer sentado até que todos terminassem e entregassem a prova. A ordem de saída da sala era a partir da fileira mais à esquerda e do candidato da primeira carteira até a última. Só podia sair um candidato por vez. Ao sair deveria ficar esperando um atrás do outro para poderem entrar na sala B (laboratório de informática)
Na sala B o processo seria o mesmo: um candidato por vez, ocupando o lugar vago (do fundo para a entrada a partir da fileira mais à esquerda). Como já era a segunda etapa da prova, assim que terminasse poderiam sair.
Implemente essas manobras do concurso seguindo as diretrizes:

(i) Defina a(s) estrutura(s) a ser(em) usada(s)
(j) Construa as rotinas (separadamente):
- entrada de um candidato (por vez) na sala A
- saída de um candidato (por vez) da sala A
- entrada no corredor para esperar o início da prova na sala B
- saída do corredor para entrada na sala B
- entrada de um candidato na sala B

(k) Construa uma rotina que, usando as rotinas acima, simule esse concurso a partir da entrada dos 30 candidatos na sala A até a entrada dos candidatos na sala B.

15) Sabe-se que, na matemática, é possível descrever uma expressão de três formas: pré-fixa, infixa e pós-fixa. Essas formas tem relação direta com o uso dos operadores (+, -, /, *).
Normalmente se usa a notação infixa, onde os operadores aparecem entre os operandos, por exemplo:

x + y
4*x
(x + 2) * z
x + 2 * z

mas pode-se usar a notação pré-fixa (operadores primeiro):

+ x y
* 4 x
*+ x 2 z
+ x * 2 z

ou pós-fixa (operadores no final):

x y +
4 x *
x 2+ z *
x 2 z * +

Faça um programa em C que permita ao usuário:

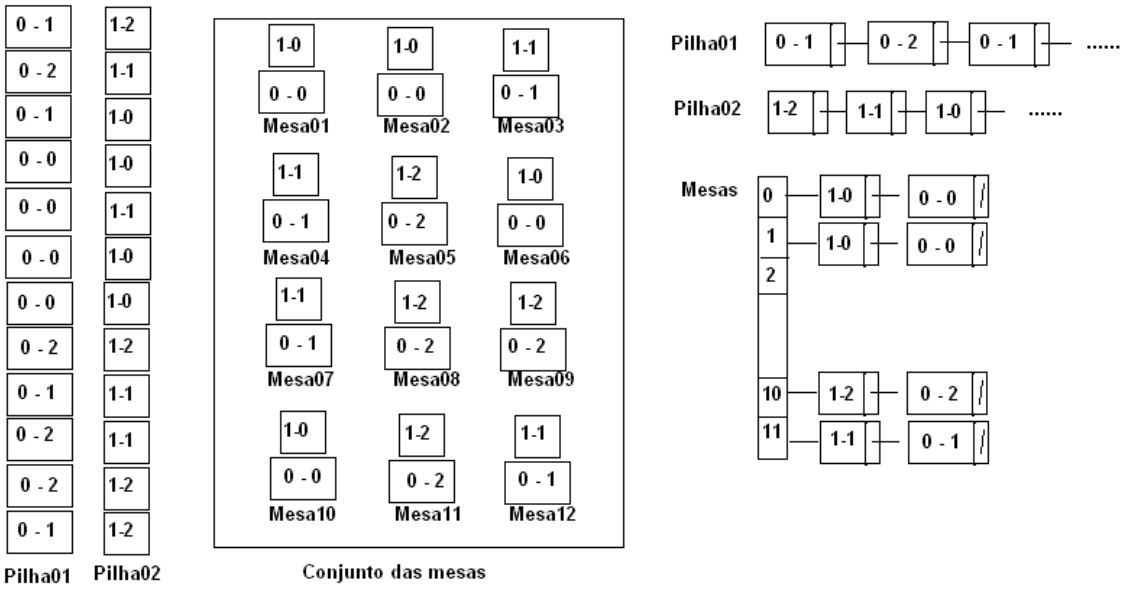
- entrar com expressões matemáticas em qualquer formato (veja os formatos descritos acima) que usa os operadores básicos (+, -, *, /)
- calcular o valor da expressão
- entrar com quantas expressões que quiser.

Todas as pilhas necessárias para o seu programa podem ser implementadas com alocação dinâmica ou estática.

16) Considere o conceito de pilha e as estruturas abaixo:

```
typedef struct pratos{
        int tamanho;         // 0-prato raso, 1-prato de sopa
        int identificação    // 0-azul, 1-verde e 2-vermelho
        struct pratos* prox;
        } *def_estrutura;
typedef def_estrutura organização_de_pratos[12];
```

Suponha que existam duas pilhas de pratos: uma com somente os pratos rasos (tamanho 0) e outra somente com pratos de sopa (tamanho 1). Nas duas pilhas existem a mesma quantidade de pratos de emblemas azuis (identificação 0), verdes (identificação 1) e vermelhos (identificação 2), só que não estão ordenados, ou seja, juntos.
O problema consiste em organizar os pratos nas mesas, onde em cada mesa deve ter um prato raso sob um prato de sopa e os pratos devem ser do mesmo emblema (azul, verde ou vermelho). Veja a figura abaixo. O primeiro número se refere ao tamanho e o segundo à sua identificação. Assim, na Mesa01 temos um prato raso azul (0-0) sob um prato de sopa azul (1-0), bem como na Mesa02, Mesa06 e Mesa10. Em termos de implementação teríamos o desenho da última coluna.



- Implemente essa ação de manipulação das pilhas, bem como a construção do vetor Mesas.
- Se preferir defina outras estruturas. O importante é que se tenha uma estrutura para guardar as pilhas de pratos e outra que se refere às mesas.

17) Tendo duas definições de pilhas como definidas abaixo, resolva os ítens abaixo.

```
typedef struct no_pilha01 {
            float dado;
            struct no_pilha01* prox;
            } *def_pilha01;          // Lista Linear Simplesmente Encadeada
typedef struct no_pilha02 {
            char dado;
            struct no_pilha02* prox;
            } *def_pilha02;         // Lista Linear Simplesmente Encadeada
```

Implemente uma rotina que leia três pilhas (duas contendo números e outra contendo char) e devolva uma quarta pilha que será a operação entre as duas pilhas de números segundo a operação armazenada na pilha que contém char (veja Figura 01). Não percorra as pilhas. Use as rotinas de empilha e desempilha.

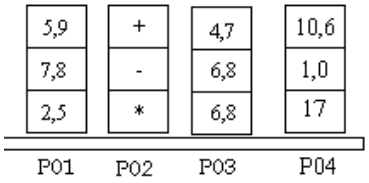


Figura 01. Exemplo de operações entre as pilhas.

Suponha que:

- As pilhas possuem a mesma quantidade de elementos (Não precisa fazer essa verificação)
- As operações permitidas são: soma(+), subtração (-), multiplicação (*) e divisão (/)

18) PROBLEMA: Eva é um robô doméstico encarregado de passar roupas num condomínio. Cada dia da semana ela vai numa casa diferente. Hoje ela vai à casa da D.Joana. D.Joana lava suas roupas e depois de

secas, dobra-as e as coloca encima de uma mesa formando uma grande pilha de roupas. Ao chegar, Eva recebe duas ordens:
(ordem 1) Primeiramente separe a grande pilha de roupa que está sobre a mesa em pilhas de tipos de peças semelhantes, ou seja, uma pilha para camisetas, outra para bermudas, outra para camisas, outra para calças e uma para vestidos.
(ordem 2) Pegue uma pilha por vez, passe as roupas dessa pilha e deixe a pilha de roupa passada ordenada por cores. Primeiro as mais escuras, depois as medianas e por cima as roupas claras.
Dadas essas instruções, D.Joana a deixa executar as tarefas sozinha.

IMPLEMENTAÇÃO: Implemente o que se pede, seguindo as diretrizes:
Considere que existem três rotinas pré-definidas para Eva:

```
int empilha (def_pilha P, struct roupa Peca);
int desempilha (def_pilha P, struct roupa *Peca);
int passar_peça(struct roupa Peca); // após passar a roupa, retorna 1
```

Onde:

```
struct roupa {
        int cod_tipo; //0-camiseta, 1-bermuda, …
        int cod_cor; // 0-clara, 1-mediana, 2-escura
};
typedef struct no{
        struct roupa R;
        struct no* prox;
} *def_pilha;
```

OBS1: Se você quiser mudar o tipo da pilha (aqui representada por uma Pilha com uso de Lista Linear Simplesmente Encadeada), por outra, pode fazer, mas redefina o que for necessário.
OBS2: Se você preferir trabalhar diretamente com cod_tipo e cod-cor dentro da definição da pilha, pode redefinir, mas o faça para tudo que for necessário.

(a) Defina a estrutura que representa a mesa que contem as pilhas de roupas recolhidas por D.Joana. Ela será a mesma onde ficarão as pilhas das roupas já passadas e organizadas por cor. Deixe explícito na estrutura as pilhas sobre a mesa, sabendo que pode variar até 10.
(b) Sabendo que Eva só reconhece as 03 rotinas acima, ensine-a, através de uma rotina, a executar as ordens da D.Joana, ou seja:

(b.1) Ordem 1 – rotina que fará a separação das pilhas das roupas recolhidas por D.Joana em pilhas separadas por tipo de peça. (Imagine que, no local, exista outra mesa que sirva de apoio para esse processo.)
(b.2) Ordem 2 – rotina que irá passar as roupas, pilha após pilha, criando pilhas de roupas passadas ordenadas por cores. (Imagine que, no local, exista uma mesinha de suporte)
(b.3) Execução total da tarefa – rotina que receberá as pilhas da D.Joana e devolverá as pilhas com as roupas passadas e ordenadas.

(19) Leia uma frase qualquer e inverta cada uma das palavras dentro da frase, usando somente PILHAS (Por exemplo: Entrada - "Ajuda teus semelhantes" Saída – "adujA suet setnahlemes"). Implemente:

(a) Defina a PILHA1 que conterá cadeias de caracteres (de máximo 50 caracteres). Essa pilha conterá a frase. Se quiser, pense que só posso ter frases com no máximo 20 palavras.

(b) Defina a PILHA2 que conterá caracteres. Essa pilha conterá uma palavra. Se quiser, pense que só posso ter palavras com no máximo 50 caracteres.
(c) Construa a rotina de Empilhamento de uma palavra na pilha do tipo PILHA1.
(d) Construa a rotina de Inversão das palavras como no exemplo acima, onde entra uma pilha do tipo PILHA1 e sai outra pilha do tipo PILHA1 com as palavras invertidas.
(e) Faça as rotinas para Desempilhar uma palavra da pilha do tipo PILHA1; Empilhar um caracter na pilha do tipo PILHA2; Desempilhar um caracter da pilha do tipo PILHA2; visualizar pilha do tipo PILHA2 e pilha do tipo PILHA1;

Considere que na função main() sejam chamadas as seguintes rotinas: (1) a leitura da frase; (2) separação das palavras e empilhamento na Pilha do tipo PILHA1; (3) CHAMADA DA ROTINA DE INVERSÃO ; (4) formação da frase a partir da Pilha do tipo PILHA1.

(20) Seja um setor do supermercado que trata somente de cervejas. Sabe-se que existem várias marcas: Skol, Brahma, Bavária, Original, Antarctica etc. Algumas em várias versões: garrafinha de 300ml, lata de 350ml, garrafa de 600ml e litrão de 1L etc.
Um funcionário resolveu cadastrar as cervejas no sistema computacional da empresa e gostaria de cadastrar as seguintes informações: marca da Cerveja (1-skol, 2-brahma etc), tipo de vasilhame (1-garrafinha 300ml, 2-lata de 350ml, 3-garrafa de 600ml, 4-litrão), quantidade no estoque. A quantidade no estoque é unitária. Sabe-se que as garrafinhas vem caixas de 24 unidades, latinhas em embalagens de 12, garrafa em caixas de 24 e litrão em embalagens de 12 unidades. Quando chega o carregamento, ele verifica as informações acima e atualiza o estoque geral da loja.
Assim que chega a carga, o funcionário confere a carga na nota fiscal, recebe a carga e atualiza o estoque geral da loja. Depois manda um funcionário da reposição colocar parte da carga na área de venda. O repositor pega cada caixa da pilha de cervejas e coloca no seu carrinho (empilha no carrinho) e registra quantas caixas está levando (assim, é criado um estoque da área de venda da loja e atualizado o estoque geral). Leva até a área correta, retira do seu carrinho e empilha no local correto.
(a) Defina a estrutura que guardará o estoque geral e da área de venda
(b) Defina uma estrutura que representará cada caixa de cerveja (pilha de cervejas na área de venda). Lembrese que precisa guardar as informações cadastradas no sistema.
(c) Construa a rotina de Empilhamento de uma caixa de cerveja no carrinho/local_adequado, lembrado de ir atualizando a quantidade na pilha, com valores unitários.
(d) construa uma rotina para desempilhamento de uma caixa de cerveja.

(21) (URI Online Judge | 1522-modificado) Claudio inventou um novo jogo, chamado de *Jogo das pilhas*, e quer submetê-lo ao próximo concurso de jogos da URI (União Recreativa Internacional). Apesar de muito divertido, o jogo parece ser muito difícil de ganhar, logo Claudio pediu sua ajuda para avaliar se algumas instâncias do jogo podem ser vencidas. O *jogo das pilhas* é individual, e é jogado com três pilhas, inicialmente com o mesmo número de cartas. Cada carta tem um valor numérico inteiro de 0 até 9. O jogador pode, a qualquer momento ver o valor de qualquer carta, mas só pode jogar com as cartas que estão no topo das pilhas. Em cada rodada, o jogador pode remover qualquer combinação de cartas que estejam no topo da pilha (pode escolher 1, 2 ou até 3 cartas) cuja soma dos valores seja múltipla de 3. **O jogo é ganho quando todas as cartas forem removidas das pilhas. Se alguma carta não puder ser removida, perde-se o jogo**.
(a) Use o tipo de pilha que preferir (estática ou dinâmica). Somente a declare.
(b) Suponha que as rotinas de empilha e desempilha já estejam prontas. Escreva somente o cabeçalho delas.

Entrada

A entrada é composta por várias instâncias. Cada instância é iniciada por um inteiro **N** ($0 \leq \mathbf{N} \leq 100$), que identifica o número de cartas em cada pilha. A entrada termina quando **N** = 0. Cada uma das **N** linhas seguintes contém três inteiros **A**, **B** e **C**, que descrevem os valores numéricos das cartas em um nível da pilha ($0 \leq \mathbf{A}, \mathbf{B}, \mathbf{C} \leq 9$). As pilhas são descritas do topo até o fundo.

Saída Modificada.

(c) Faça uma rotina para que ao receber os dados de entrada (N e as três pilhas), retorne 1 (se o jogador pode ganhar) ou zero (caso contrário). Considere que sempre será retirada uma carta de cada uma das três pilhas de cartas. Se não usar as três cartas, devolva-as às pilhas correspondentes.

(22) (URI Online Judge | 1438-modificado) Joãozinho e sua família acabaram de se mudar. Antes da mudança, ele colocou todos os seus livros dentro de várias caixas numeradas. Para facilitar a retirada dos livros, ele fez um inventário, indicando em qual caixa cada livro foi colocado, e o guardou na caixa de número 1. Chegando no seu novo quarto, ele viu que seus pais guardaram as caixas em várias pilhas, arrumadas em fila, com cada pilha encostada na pilha seguinte. Joãozinho é um garoto muito sistemático. Por isso, antes de abrir qualquer outra caixa, ele quer recuperar seu inventário. Joãozinho também é um garoto muito desajeitado. Para tirar uma caixa de uma pilha, ele precisa que a caixa esteja no topo da pilha e que ao menos um de seus lados, não importa qual, esteja livre (isto é, não tenham nenhuma caixa adjacente). Para isso, Joãozinho precisa desempilhar algumas das caixas. Como o quarto dele é bem grande, ele sempre tem espaço para colocar as caixas retiradas em outro lugar, sem mexer nas pilhas que os pais dele montaram. Para minimizar seu esforço, Joãozinho quer que você escreva um programa que, dadas as posições das caixas nas pilhas, determine quantas caixas Joãozinho precisa desempilhar para recuperar seu inventário.
(a) Use o tipo de pilha que preferir (estática ou dinâmica). Somente a declare.
(b) Suponha que as rotinas de empilha e desempilha já estejam prontas. Escreva somente o cabeçalho delas.

Entrada

A entrada é composta de vários casos de teste. A primeira linha de cada caso de teste contém dois números inteiros **N** e **P** , indicando, respectivamente, o número de caixas e o número de pilhas ($1 \leq \mathbf{P} \leq \mathbf{N} \leq 1.000$). As caixas são numeradas seqüencialmente de 1 a **N**. Cada uma das **P** linhas seguintes descreve uma pilha. Cada linha contém um inteiro Qi, indicando quantas caixas há na pilha i, seguido de um espaço em branco, seguido de uma lista de Qi números, que são os identificadores das caixas. Os elementos da lista são separados por um espaço em branco. Todas as pilhas contêm pelo menos uma caixa, e todas as caixas aparecem exatamente uma vez na entrada. As caixas em cada pilha são listadas em ordem, da base até o topo da pilha. Todas as caixas têm o mesmo formato. O final da entrada é indicado por **N** = **P** = 0.

Saída Modificada.

(c) (2,0pt) Faça uma rotina para que ao receber os dados de entrada e calcule o número mínimo de caixas, além da caixa 1, que Joãozinho precisa desempilhar para recuperar o seu inventário.
Por exemplo:

4 3 1 3 2 1 2 1 4	Saída = 2.