1)

```
void troca(int *a, int *b) { // valores passados por referência
      int aux;
      aux = *a;
      *a = *b;
      *b = aux;
}
```

Exemplo de chamada:
Dado dois inteiros a e b: troca(&a, &b);

2) As duas funções servem para alocar dinamicamente (em tempo de execução) um bloco de memória, retornando um ponteiro para o primeiro endereço alocado. A diferença principal é que a função malloc() não se preocupa com os conteúdos dos endereços, assim eles podem conter algum lixo de memória. A função calloc() limpa o conteúdo dos endereços, atribuindo-lhes o valor 0.

3) O aluno não conseguiria cadastrar e imprimir corretamente. Como a estrutura está no arquivo Livro.c (seguindo o conceito de encapsulamento), o aluno não conseguiria declarar um tipo Livro de forma direta:
Livro l;
ele precisaria declarar:
Livro *l;

Sendo l um ponteiro, o acesso a seus campos mudaria. Em vez de:
l.edicao = edicao;
teríamos:
l->edicao = edicao;
ou:
(*l).edicao = edicao;
por exemplo.

Além disso as lógicas para cadastrar e imprimir deveriam estar contidas em funções no arquivo Livro.c. Em um TAD só é possível interagir com a estrutura por meio de funções, não diretamente.

4)
a)
Lista: estrutura que guarda uma sequência de itens. Permite inserir/remover itens em qualquer posição (início, meio, fim).
FIla: tipo de lista onde as inserções são feitas sempre no fim e as retiradas sempre no início. Seguindo a ideia de que o primeiro que chega é o primeiro a sair, como em uma fila de banco.
Pilha: tipo de lista onde as inserções e retiradas são sempre feitas no topo. Seguindo a ideia de que o último que entra é o primeiro que sai, como em uma pilha de pratos.

b) Poderia ser resolvido com qualquer uma delas. Uma maneira bem intuitiva seria com fila, visto que pegamos o primeiro elemento e comparamos com todos os outros após ele. Depois pegamos o segundo elemento e comparamos com todos os outros após ele (perceba que o primeiro já não é útil e pode ser descartado). Isso nos dá a ideia de uma fila, já que sempre precisamos usar o primeiro elemento.

Assim, uma ideia seria seguir o passo a passo abaixo:

- criar uma fila vazia
- para cada item na sequência:
    - enfileirar item
- enquanto a fila não estiver vazia:
    - desenfileirar primeiro
    - comparar primeiro com o próximo item até encontrar um item maior ou então até chegar ao final da fila
    - se encontrou um maior
        - imprimir maior
    - se não
        - imprimir -1

Também poderia ser feito com pilha. Considerando o exemplo [4, 5, 2, 25]:
Primeiro item da sequência: 4, como a pilha está vazia:

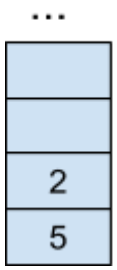
- empilhar o 4

Segundo item da sequência: 5. Como 5 > 4, logo o maior do 4 já foi encontrado, então:

- imprimir 5 e 5
- desempilhar 4 (ele não é mais necessário)
- empilhar o 5

Terceiro item da sequência: 2. Como 2 < 5, ainda não foi encontrado o próximo do 5, então mantemos o 5 e guardamos o 2:

- empilhar 2



Quarto e último item da sequência: 25. 25 > 2, logo o maior do 2 já foi encontrado, então:
- imprimir 2 e 25
- desempilhar 2

25 também é maior que 5, quer dizer que o maior do 5 foi encontrado, então:
- imprimir 5 e 25
- desempilhar 5

A pilha está vazia agora, então:
- empilhar 25

O 25 sobra na pilha, quer dizer que ele não tem próximo, então:

- imprimir 25 e -1
- desempilhar 25

Um possível passo a passo para resolver o problema com pilha:

- para cada item da sequência
    - se pilha está vazia ou item < topo
        - empilhar item
    - se não
        - enquanto item > topo
            - imprimir topo, item
            - desempilhar
        - empilhar item
- enquanto pilha não for vazia:
    - imprimir topo, -1
    - desempilhar

5) Baseado no exercício da lista:
https://github.com/emanoelim/AE22CP/blob/master/04-15%20-%20Exerc%C3%ADcios/exerc%C3%ADcios_pilhas_filas.md

Primeira pergunta: O aluno A guardou 3 itens, enquanto o aluno B guardou 5.

Segunda pergunta: O aluno A joga o final da fila sempre para a direita:
f->ultimo++;

Ele faz o mesmo com o início da fila:
f->primeiro++;

Isso quer dizer que a fila dele só caminha da esquerda para a direita. Assim as posições que vão sendo liberadas à esquerda nunca mais são ocupadas e como poucas operações se atinge o limite da fila.

O aluno B caminha na fila de forma circular:
f->ultimo = (f->ultimo + 1) % MAXTAM;
f->primeiro = (f->primeiro + 1) % MAXTAM;

Com essa operação, se o último item da fila é 4 e o MAXTAM é 5, por exemplo, ao adicionar mais um item, o último passa a ser:
ultimo = (4 + 1) % 5
ultimo = 5 % 5
ultimo = 0

Ou seja, as posições liberadas mais a esquerda conseguem ser aproveitadas.

Terceira pergunta: A implementação do aluno B é mais vantajosa por aproveitar todas as posições da fila, diferente da implementação do aluno A, onde tempos posições que ficam inutilizáveis.

6) Para resolver este problema é preciso:

a) Saber se a bagagem estourou o limite de peso, então poderia ser criada uma função que recebe a lista e o peso da mala e retorna o peso total da bagagem.
b) Saber qual é o item mais pesado, para poder removê-lo. Sabendo que remoções em uma lista implementada por meio de vetores são feitas com base no índice, podia ser criada uma função que recebe uma lista, encontra o item mais pesado e retorna seu índice.
c) Remover o item mais pesado, dado seu índice. Seria necessário criar uma função que recebe um índice e removo o item correspondente.
d) Criar uma função principal contendo um loop que seria executado enquanto o peso da bagagem fosse maior que o permitido. Nesse loop buscaria o índice do item mais pesado fazendo sua remoção.

Obs.: esta é uma sugestão de solução, existem várias linhas de raciocínio para resolver.