1. Empilhar: Vc da ++topo e coloca o valor dendro do seu devido local do array assim armazenando ele e fazendo ele ser o primeiro da pilha.
   Desempilhar: Basicamente vc da topo— e não precisa ligar para o conteúdo que ainda está na posição, quando vc empilhar novamente ele sobrescreve o conteúdo anterior.
2. Uma lista simplesmente ligada é mais leve na memoria e custa menos locação de memória, mas quando comparado a lista duplamente encadeada ele não consegue fazer o percurso de volta na lista, fazendo que dependendo da situação que vc deseja percorrer a lista a duplamente terá uma vantagem.
3. Bom, uma lista encadeada tem uma vantagem principalmente em relação a memoria que é uma lista onde os item não precisam ser alinhados em sequencia, fazendo que ele seja muito mais fácil de se adaptar a memoria atual comparado a um array que so pode alinhar os itens em sequencia, tem o fato que a lista tem o tamanho modulável e o array tem que ter seu tamanho definido na declaração.
   Complemento:
   No entanto, o acesso aos elementos em uma lista ligada é mais lento, pois requer travessia a partir do início da lista, enquanto em arrays o acesso é imediato via índice. Além disso, implementar operações em listas ligadas exige gerenciar cuidadosamente os ponteiros para evitar *memory leaks* ou acessos inválidos.
4. Podemos acessar um ponteiro de uma struct de 2 formas: p->x ou (*p).x.
   O principal fator do ponteiro é simplesmente se bem utilizado vc não liga para escopo e pode referenciar o valor de onde quiser.
5. O ponteiro é justamente a ulilização de um espaço de memoria que referencia outro espaço de memoria, assim podendo acessar o dado desse espaço e referenciar de forma indireta. O uso essencial para um ponteiro seria justamente vc precisar alterar/retornar dois valores de uma função ou quando vc faz o uso de um novo tipo de dado.
6. Structs em C faz justamente a criação de um novo tipo de dado abstrado, permitindo assim que dados possam andar de forma junta e unida e não serem separados, tambem ajudando na organização do código na modularização do mesmo e permitindo a similaridade de um pseudoobjeto no C, tambem vale ressaltar a continuidade na memoria, de foma que quando uma struct é instanciada na memoria ele cria seus componentes em sequencia. Um exemplo seria uma struct de aluno: int Macricula, char Nome[100] e etc.
7. Malloc serve para criar um ponteiro para um tipo de dado escolhido pelo dev na heap, assim podendo ser referenciado em qualquer local se quiser e podendo escolher seu tamanho tambem. Uma boa pratica é verificação

pos alocação da memoria para ver se ele não alocou um local que possuía lixo de memoria. Vale falar tambem a presença do free para liberar esse ponteiro e tambem aponta-lo para Null.

8. Entendendo que existe uma lista já com os dados, eu criaria 2 novas listas de ponteiros onde quando cada termo da lista original passa-se por uma verificação de par, a lista par teria um ponteiro para esse conteúdo e se não passasse a impar apontaria para esse conteúdo e assim por diante para cada elemento, assim armazenando os dados em sua devida lista sem duplicar o valor do nó.
9. A fila segue o padrão de FIFO ou seja primeiro a entrar primeiro a sair, seguindo esse ideia ela teria uma struct onde marcaria o começo,fim,tamanho e seus itens, através das funções quando alguém saísse do começo da fila o começo passaria para frente e se alguém entrasse o fim passaria para frente. Uma inserção de um novo dado nada mais é que a passar o fim para 1 valor a mais e alocar o dado. Fila: FIFO e pilha: LIFO
10. Quando estamos falando de inserir uma novo nó dentro da função criamos o mesmo e declaramos o valor dentro dele, fazemos a verificação se ele foi criado corretamente e não está alocando lixo, depois falamos que ele aponta para null. Já na nossa lista nos pegamos ela vamos até o ultimo elemento e colocamos o ultimo Nó da lista referenciando o nó criado, assim adicionado o novo nó na lista. É importante ajustar justamente para não perder a referencia de nem um nó, assim tendo memoriy leak caso algo de errado.