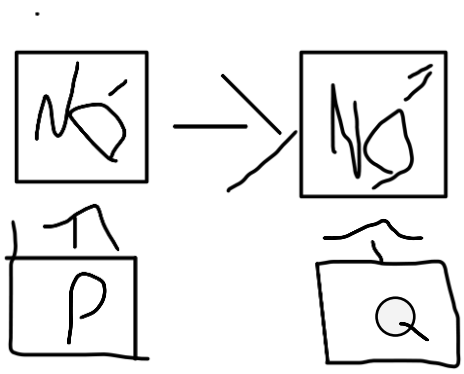
1. Tipos abstratos são tipo criados por uma structs, eles são tipos que não são originais do C mas através da sua criação permitem ser manipulados tendo seus atributos e permitindo que os dados andem juntos e sejam alocados em sequencia na memoria. Uma de suas vantagem a Abstração é a ideia de poder ter vários atributos dentro de uma só variável por exemplo e manipula-lo quando nescessario, ajudando tambem a criar estruturas não padrões em C como pilhas e filas dependendo da sua manipulação. Um exemplo de pilha é que depois que a struct foi criada e tem suas funções declaradas qualquer pessoa com um conhecimento básico pode manupular esse dado abstrato como um dado originário de C, criando diferentes aplicações como a torre de hanoi ou um jogo de baralho por exemplo.
2. Typedef struct{
   Int dados[10], começo, fim, tamanho;
   }fila;
   Funções:
   peak: Pega o primeiro elemento e retorna ele.
   Dequeue: tira o primeiro elemento da fila e retorna ele.
   Enqueue: Adiciona um novo elemento ao fim da fila.
   IsEmply: Verifica se está vazia
   IsFull: Verifica se está full
   As operações devem se modular no sentido de por exemplo na hora de adionar um novo elemento se ele estiver no final ele passa a ser o inicio de novo (caso não estiver cheio) , assim como remover onde o começo deve ser resetado se chegar no fim. Uma fila circular é melhor quando se usa uma fila de forma mais prolongada pois ela vai se adaptando e se adequando, enquando caso a fila convencional for muito utilizada chega uma hora onde ela não funcionara mais.
3. O ponteiro é essencial em usos como arvores e listas para percorrer os elementos e indicar o fim deles. O ponteiro nada mais é que um endereço que guarda outro endereço de memoria podendo de forma indireta acessar o valor do endereço guardado e referencia-lo. Um uso errado de ponteiro é em situações onde vc declara um ponteiro e esquece de dar free nele assim fazendo alocamento de uma memoria em seu combutador até ele ser desligado, ou quando vc esquece de fazer a verificação do local que foi alocado podendo apontar para lixo.
4. Quando definimos que p =&q entendemos que p passou a apontar para q e de certa forma passou apontar para o nó que q estava apontando. Eles podem afetar que perderam o referenciamento de um dos nos que era apontado pelo p anteriormente e de certa forma em remoções da lista

assim como inserções podem ser realizadas em um local errado caso a rota tenha sido quebrada



5. Quando damos um push em uma pilha nos entendemos que o topo dela será inclementado e que o valor passado será atribuído a localização do array referente ao topo, já no pop entendemos que o o topo será reduzido e o valor se manterá so não será mais referenciado. Lembre de levar em relação o LIFO. A memoria Stack leva em consideração o modelo de pilha justamente porque quando estamos chamando uma função esperamos ela terminar de ser executada finaliza-la e quando estamos fazendo recursão chamamos ela varias vezes de modo que quando a ultima a ser executada acontecer suas anteriores poderão terminar, seguindo assim o modelo de LIFO. Um exemplo é fatorial onde ele chamara ela mesmo diminuindo de um em um ate dar 1 e depois retornará multiplicando os resultados.
6. O conceito de lifo é evidenciado durante a remoção do elemento de forma que o ultima a entrar será o primeiro sair assim de forma que somente o 7 e o 9 permaneceram na fila.
7. A função isEmpty(pilha *p){

```
        If(*p == NULL){
                Return 1;
        }else{
                Return -1;
        }
}
```

   Temos que entender que quando estamos falando de uma lista mesmo que esteja simulando uma pilha para verificar se está vazia ela deve ter seu valor Null representando que não possui nem um nó vinculado a ela. Ela inporante justamente para entender que principalmente em relação a ordem de execução de uma expressão matemática onde vc monta uma pilha de operadores para realiza-las de uma ordem certa, assim a função ira verificar se a pilha de operadores acabou.
8. A fila FIFO e pilha LIFO tem intuídos diferentes de ordem onde dada um segue um modelo, no caso da FIFO ele da prioridade na saída aquele que

entrou primeiro e no caso da LIFO a aquele que entrou por ultimo. Fila de banco por exemplo é um exemplo de fila, pois o primeiro que chegou na fila será o primeiro a ser atendido, e de pilha um baralho empilhado onde vc vai retirando as cartas de cima ou de pratos, onde o topo no caso o ultimo que entrou é o primeiro a sair.

9. No caso a fila terá os elementos a partir da 1 colocação co array e tendo os elementos 10,15,20.
10. Void Dequeue(fila *f){

    If(f->nó->nó != Null){

        f->nó = f->nó->nó

    }

    }

    Ela deve ter cuidado para não perder a referencia do inicio da fila e nem do resto do conteúdo de modo que ele não seja perdido e além disso de modo que conforme menos conteúdo exita na fila mais rapido percola fica
11. A diferença é justamente que a lista simplimente encadeada percorre de forma unitamente direcional,todavia é mais leve. Já na dulplamente encadeada a lista percorre para ambas as direções de modo que é mais rapido para a localização de um nó na lista embora consuma mais memoria. A melhor forma de comparar desempenho é verificar o ultimo nó em ambas as listas sendo circulares onde a simplesmente deve percorer toda a lista para chegar o ultimo e a dulplamente deve percorer 1 nó do começo para chegar no ultimo.
12. Deve ser criado um nó que com o valor 4 e percorrer o vetor ate chegar no nó 3, de modo que o nó com o valor 4 irá apontar para o vetor de valor 5 e o vetor de valor 3 irá passar a apontar para o de valor 4.
13. A vantagem é justamente em referencia a percorrer a lista. Funções como procurar elementos ou pegar um elemento em uma localização especifica se beneficiam muito do fato de somente poderem seguir por caminhos mais curtos e chegar no nó de forma mais rápida, de modo que em relação a simplesmente encadeada encontrar um elemento obrigaria a percorre-lo todo independente da situação.
14. No caso o ponteiro 4 passaria a direcionar o 8 e o ponteiro anterior do 8 indicaria a o 4 de modo. A ideia é a manipulação do nó em referencia do 6 então o nó 8 passaria a apontar como seu antecessor o 4 sem tirar o 6 ainda da lista e depois se referenciando no 4 o próximo seria o 8, assim o nó 6 não estaria mais na ordem estando a parte e somende esperando para ser liberado da memoria. De forma que nem uma referencia da lista principal seja perdida assim funcionando corretamente.

15.
```
void deleteNode(No *nodo) {
    if (nodo->anterior != NULL)
        nodo->anterior->proximo = nodo->proximo;
    if (nodo->proximo != NULL)
        nodo->proximo->anterior = nodo->anterior;

    free(nodo);
}
```