Struct e Cabeçalhos de Funções

O primeiro passo é a struct, vamos de Node (cada elemento de uma pilha é chamado de nó).
Ela vai ter apenas dois elementos, um número inteiro e um ponteiro para outra struct do tipo Node.

Esse ponteiro do próprio tipo é obrigatório, não existe estrutura dinâmica de dados sem esse detalhe especial. Na lógica do funcionamento da pilha, vamos entender para que ele serve.

Já o outro elemento (**int num**) é só para armazenarmos números nessa pilha, pois vamos pedir e mostrar esses números. Mas note que isso é uma struct, podemos colocar quantos elementos e do tamanho que quisermos, que a pilha vai funcionar do mesmo jeito.

Também colocamos todos os cabeçalhos das funções que iremos usar para programar a pilha em C, por questões de organização (o código das funções ficará abaixo).

Função main()

Na função main criamos a nossa pilha, que é uma struct Node, vamos chamar ela de "PILHA", e é a base. Quando o primeiro elemento for adicionado, ele vai ser adicionado no ponteiro "prox" da PILHA.
Este nó, na verdade, não faz parte da pilha, ele serve apenas para indicar onde ela começa (começa no ponteiro na qual ela aponta).

O resto da main é simplesmente um laço do while, que fica exibindo um menu com as opções para trabalharmos com a pilha, que termina se digitarmos 0.

Função menu() e opcao()

Essas são as funções responsáveis pela ação, a interação o usuário e a pilha.

A função menu() simplesmente exibe as opções possíveis e pede um inteiro ao usuário. Este inteiro será usado e passado para a função opcao(), que junto com a pilha (ponteiro *PILHA) vai servir para chamar a função específica, de acordo com o que o usuário escolheu.

Na função opcao(), basicamente existe um switch que vai tratar a opção escolhida pela usuário, e chamar a função correta. Sem segredo.

Função inicia()

Esta função é responsável por inicializar a pilha.
Inicializar é simplesmente preparar a pilha para ser utilizada, simplesmente apontando seu ponteiro *prox para NULL.

Essa função é chamada automaticamente no início de nosso programa em C, e quando zeramos a pilha.

Função vazia()

Esta função simplesmente checa se a pilha está vazia ou não.
Basta olhar para onde aponta a base (*PILHA), se apontar para NULL é porque ela está vazia, senão, é porque existem nós nesta estrutura de dados dinâmica.

Função aloca()
Visando deixar nosso algoritmo bem feito, estruturado e organizado, é interessante separar cada ideia em uma função diferente, com um propósito bem evidente e único.

A função aloca() é um exemplo dessa organização.
Como o nome sugere, ele serve para alocar nós.
Sempre que formos adicionar um elemento na pilha, temos que alocar memória para ele.

Essa função aloca a memória necessária pro nó (struct Node), pede o número que o usuário quer armazenar) e retornar o endereço da memória alocada.

Função libera()
Tão importante quanto alocar memória para cada nó da pilha de nossa estrutura de dados, é liberar esse espaço de memória. A função libera faz isso, vai liberando o espaço alocado de cada nó de nossa pilha.

Usamos dois ponteiros para a struct do nó, o ponteiro que aponta para o elemento atual e o ponteiro que aponta para o próximo elemento.
Pegamos o ponteiro que aponta para o nó atual e usamos para desalocar aquele espaço de memória, em seguida o ponteiro que apontava para o atual aponta para o próximo, e isso segue até o fim da pilha, desalocando cada um dos nós da estrutura de dados.

Função exibe()
Essa é a função responsável por exibir todos os elementos da pilha.
Como em cada nó dessa estrutura de dados possui um inteiro que o usuário inseriu, essa função, no fim das contas, vai exibir os números da pilha.

Essa função declara um ponteiro que vai começar apontando para o primeiro elemento da pilha, exibe o número armazenado ali, pega o endereço do próximo nó, exibe o que está armazenado nele também, e assim se segue, até o fim da pilha (quando *prox aponta para NULL).

Função push()
Agora vamos a parte que mais interessa em se tratando de estrutura de dados, e especificamente, sobre pilhas em C: as funções push e pop.

Push em inglês é empurrar, vamos empurrar, colocar um elemento, um nó na pilha.
O primeiro passo é alocar espaço para esteve novo nó da pilha, o que é feito com ajuda da função aloca().
Como é uma pilha, seu último elemento (que é esteve novo), deve apontar para NULL, pois isso caracteriza o fim da pilha.

Adicionado o elemento, vamos procurar o último elemento da pilha.
Temos o ponteiro *PILHA que aponta para a base.
Se a pilha estiver vazia, ótimo! Fazemos o ponteiro *prox apontar para esteve novo nó, e tudo ok.

Se a pilha não for vazia, vamos achar o último elemento através de um ponteiro *tmp que vai apontar para o primeiro elemento da pilha (PILHA->prox aponta para o primeiro nó).

E como sabemos que o nó atual é o último?
Basta checar seu ponteiro *prox, se ele apontar para NULL, ele é último.
Se não apontar, é porque aponta para um novo nó, então fazemos nosso ponteiro *tmp apontar para este novo nó, sempre, até chegar no último.

Quando "tmp->prox" apontar para NULL, é porque *tmp está apontando para o último nó.

Agora, vamos fazer o próximo nó apontar para nosso novo nó: tmp->prox = novo
E pronto! Função push feita! Adicionamos um novo nó na pilha.

Função pop()

Agora vamos para a função pop, o outro pilar da estrutura de dados dinâmica que é a pilha.
Esta função vai tirar o último nó da pilha, e retirá-lo de lá.

Primeiro fazemos uma checagem se a pilha está vazia (PILHA->prox aponta pra NULL). Se estiver, não há nada a ser feito, pois não há nó para ser retirado da pilha.

Do contrário, vamos utilizar dois ponteiros para struct Node, o "ultimo" e o "penultimo". Basicamente, o que vamos fazer é que o "ultimo" aponte para o último elemento da pilha e o "penultimo" aponte para o último nó da pilha.

O motivo disso é simples: vamos retornar o último nó da pilha e vamos retirá-lo da lista (então ele vai se perder, por isso precisaremos sempre do penúltimo, pois este vai se tornar o novo último nó da lista).

O que vamos fazer é buscar o último nó (que é aquele que tem o ponteiro *prox apontando pra NULL).
E sempre que avançarmos na pilha com o ponteiro "ultimo", fazemos com que o "penultimo" também avance (ora, o penúltimo nó é aquele que tem o ponteiro *prox apontando para o ponteiro *ultimo).

Essa lógica é feita testando ultima->prox, quando não for NULL, o ponteiro "penultimo" passa a ser o "ultimo" e o "ultimo" vai ser o "ultimo->prox", que é o próximo nó da pilha.
Note que agora que demos um passo a frente na pilha, com os dois ponteiros.
E isso só para quando estivermos apontando para o último e penúltimo nó da pilha.

Quando estivermos nesse ponto, fazemos "penultimo->prox" apontar para NULL, pois vai caracterizar que o penúltimo nó será, agora, o último nó (pois aponta pra NULL), ou seja: retiramos o último nó da pilha!

E o que fazemos com o último nó?
Vamos retornar ele! Se estamos tirando ele da pilha, é porque queremos o que tem nele, seja lá pra que for. Então retornamos ele pra função que o chamou (a função opcao()), ela exibe o valor desse último elemento da pilha e então o descarta (liberando a memória dele).