Por Andre Giolito:

**Explicação da solução do exercício Locations resolvido em C**

**PARTE I: Conhecendo o problema**

No ínicio da descrição do problema já se menciona uma possível relação de hierarquia entre as localidades e as sublocalidades, assim como no exemplo dado do Estado de São Paulo e seus municípios e bairros. A hierarquia de dados pode ser facimente implementada utilizando uma árvore, mas ainda assim resta saber qual tipo dessa estrutura devemos usar. O texto menciona que uma localidade pode abarcar quantos locais filhos quiser, então já de cara é descartada a possibilidade de uso de árvores binárias, restando então a implementação por árvores n-árias, que se assemelha muito bem com a descrição e estrutura do problema passadas.

**PARTE II: Funções, bibliotecas e estrutura**

No início do código, criou-se uma estrutura que irá simular nosso local. A estrutura contém um nome (cep) e dois ponteiros – um para o próximo local e outro para o último sublocal.

A implementação foi subdividida em uma série de funções, que além de desafogar a função *main,* deixa o nosso código mais legível. Se tratando de uma árvore, já se pode esperar uma série de funções básicas para este tipo de estrutura: busca, altura, número de nós (ou locais, como no nosso problema). Outras funções básicas foram criadas para facilitar nossas vidas na hora de criar um local, como em *criaLocal()* e *insere(),* que serão explicadas melhor na parte três*.* Outras foram pensadas para resolver problemas mais específicos de formatação e forma, como em *ordena()*, *copia()* e *imprime()*. As variáveis constantes *limite* e *numNos* definem, respectivamente, o limite de altura que é passado no problema e o número de nós que o usuário deseja colocar, mas já está definido como padrão para o tipo de output passado no texto como exemplo.

É relevante mencionar que as bibliotecas *string.h* e *stdio.h* foram utilizadas para manipular *strings* e *input/output*, respectivamente. *Malloc.h* foi escolhida para alocar dinamicamente os espaços de memória.

**PARTE III: Solução e o papel das funções na resolução**

No início da função *main,* algumas variáveis e arrays foram criadas para que possamos guardar certos valores e strings temporariamente. O *scanf()* é utilizado de modo à capturar o input passado pelo usuário e guardar nos arrays de char *destino* e *local.* Note que o nome já diz muito sobre seu funcionamento, em que *local* será uma sublocalidade de *destino*. Note também que é criado um ponteiro “m” para localidade que armazenará um endereço que será retornado pela função *crialocal().* É a única vez que chamaremos esta diretamente, pois “m” será nossa localidade raiz, assim como o “Estado de São Paulo” do exemplo.

Em seguida, a função scanf() é chamada seguidas vezes à depender da quantidade de nós, e a função *insere()* irá direcionar o local ao destino correto. A funçãocomeça chamando *níveis()*, que retorna a altura da árvore (que não deve ultrapassar o limite 100). Se a condição for satisfeita, então será feita uma busca que retorna um ponteiro para a origem do local. A busca percorre todos os locais recursivamente, compara as strings e retorna um ponteiro para local, caso este seja encontrado. Se a busca for concluída com sucesso, será feita uma chamada da função criaLocal() e uma série de troca e ajuste de ponterios. O ponteiro *sub* sempre irá indicar o ultimo local de uma hierarquia abaixo, e o *prox* para um local irmão, pertencente à mesma hierarquia.

Logo após concluirmos os locais, iremos formatar e rearranjar os locais de acordo com a ordem lexicográfica. Para este efeito, criamos um array de strings *array* que armazena todos os locais da árvore, a partir da chamada de *copia()*. Após copiarmos, devemos ordenar as strings do array chamando *ordena()* que compara strings uma a uma utilizando *strcmp(),* que devolve um número maior que zero, caso o valor numérico ASCII de uma string seja maior que a de outra, e faz a troca. Após a ordenação, chamamos *imprime()* que formata a saída e ainda imprime o número de sublocais atrelado a um nó pelo valor retornado da função *numeroLocais().* Esta ultima função percorre recursivamente os nós a partir de uma raiz dada e retorna + 1 a cada chamada em que há um nó de fato.