Arthur H. R. Carvalho – 1770

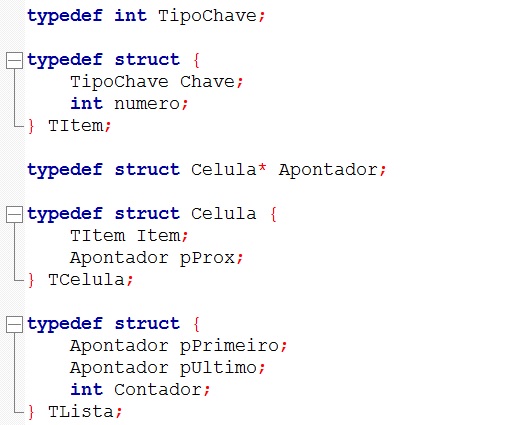
Samuel J. S. Tavares – 2240

**Introdução e objetivo**

O trabalho prático de AEDS I consistia em fazer um Tipo Abstrato de Dados (TAD) Lista Linear, em sua versão implementada por Lista Encadeada. O objetivo foi com que aprendêssemos a manipulação correta das operações e o uso devido de ponteiros, aprendendo assim a lidar com os possíveis erros que pudessem ocorrer .

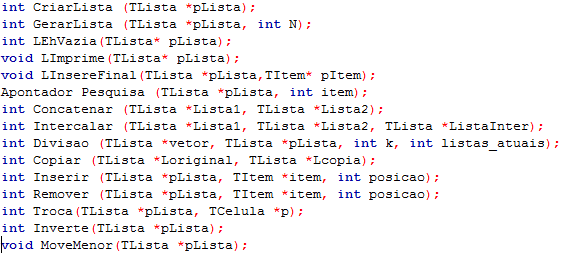
Foi dividido da seguinte forma:

Primeiro, foi feito um arquivo chamado Lista.h que continha as estruturas necessárias para a lista encadeada.



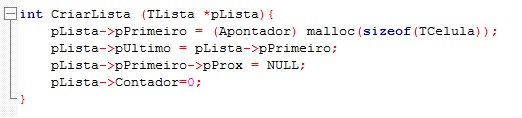
O contador, presente em TLista, foi usado para sabermos o tamanho da lista, e, em diversas operações, isso nos foi útil.

Depois, foram definidos os cabeçalhos das operações



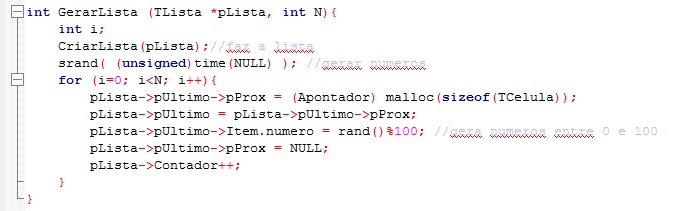
Então, começando o arquivo Lista.c pelas operações mais básicas

**Cria Lista**



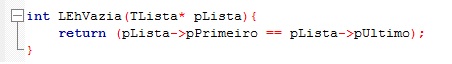
Função que cria uma lista encadeada com cabeça.

**Gerar Lista**



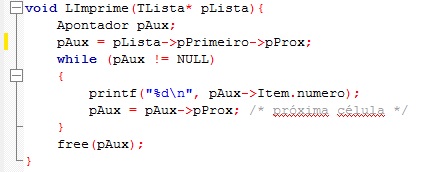
Função usada para gerar uma lista de tamanho N, com números aleatórios de 0 a 100. Esta função foi a mais usada para o teste durante o trabalho, levando em conta a praticidade na hora dos testes. A função rand “sorteia” elementos.

**Lista é vazia**



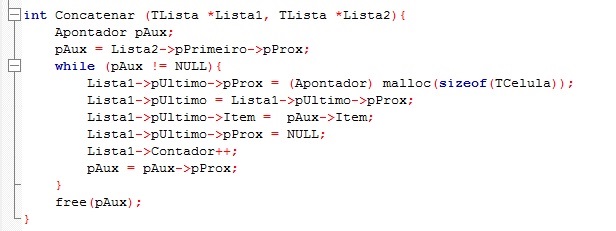
Função básica para testar se a lista encadeada está vazia. Ela retorna se o endereço de memoria da primeira posição for igual a ultima posição.

**Imprime Lista**

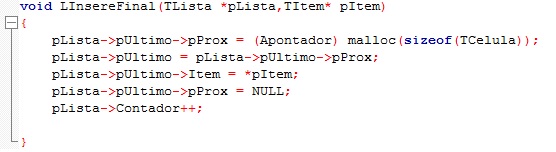
****

Função simples para imprimir a lista na tela. Nela vemos um comando while, que enquanto o pAux (um auxiliar local) for diferente de 0, ela percorre a lista exibindo cada valor nele contido.

**Concatenar**

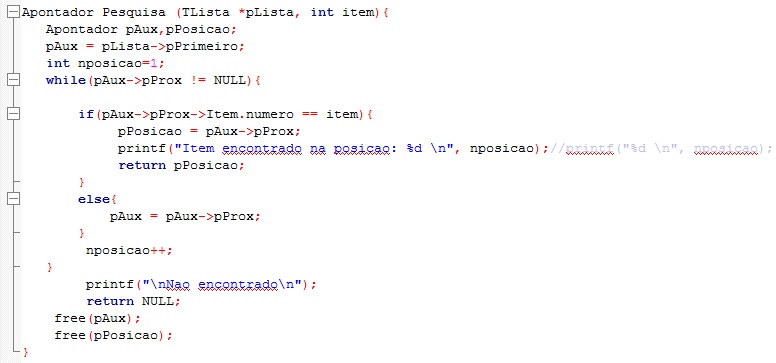
****

Função simples que “une” duas listas. Foram recebidas duas listas como parâmetro. Então, usando o pAux, um auxiliar local que apontava para o endereço de memoria da segunda lista, então, foi gerado um contador que enquanto o pAux fosse diferente de NULL, a segunda lista iria ser anexada no final da primeira, concatenando assim, as duas listas.

**Inserir ao final da lista**

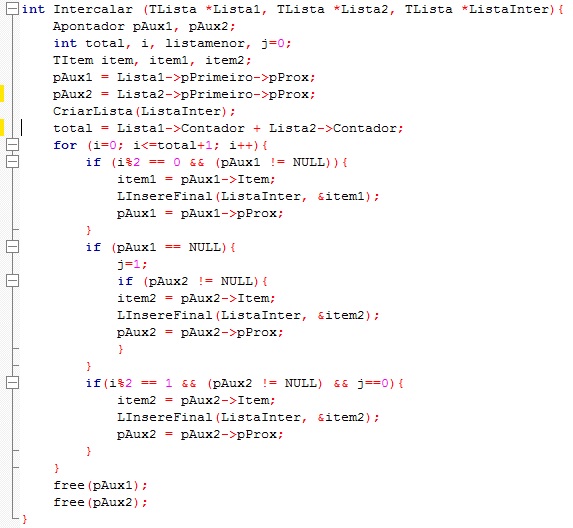
Essa função foi feita separa pois foi usada em outras. Operação simples que recebe um item e anexa ele ao final da lista.

**Pesquisar em uma lista**

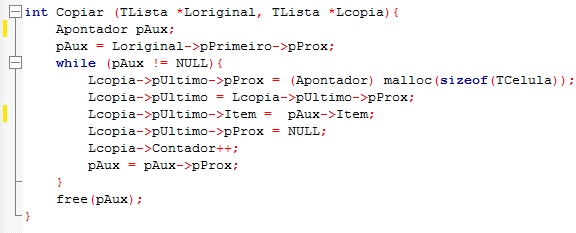


Operação que pesquisa itens específicos na lista. A pessoa escolhe qual lista e qual numero ela quer ver se está na lista, se tiver na lista, o programa retorna a posição que o numero encontra. Se não tiver na lista, ela retorna NULL.

**Intercalar listas**

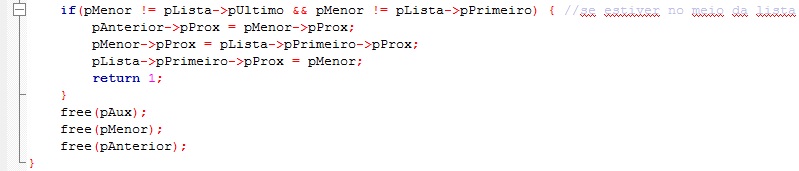
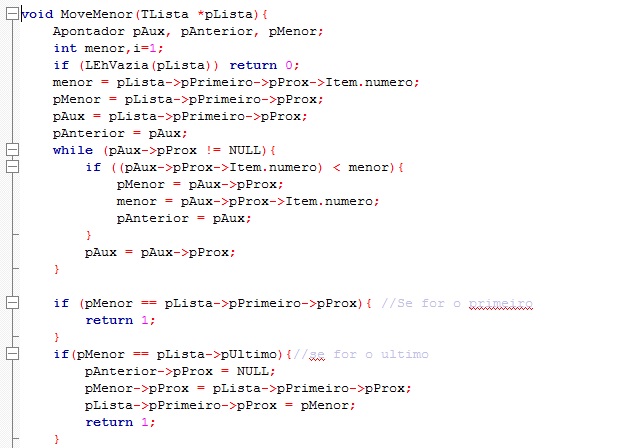
****

Recebe como parâmetro três listas. Duas delas contém números, a terceira é o resultado da operação entre as duas. Com a ajuda de dois apontadores auxiliares (onde cada um recebe o endereço de uma lista); Então é executado um comando de repetição que testa e intercala as duas listas. Ao final, foi apagado os apontadores locais usando a função free.

**Copiar uma lista em outra **

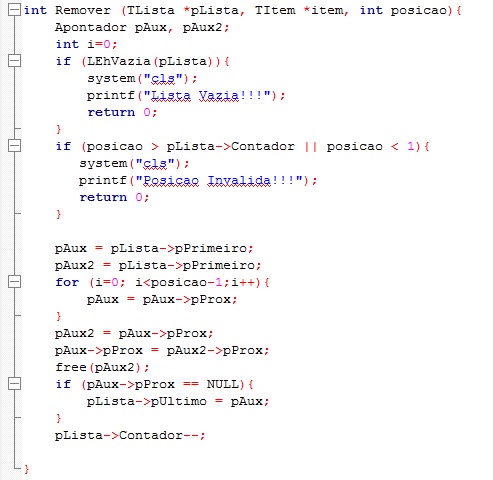
Recebe como parâmetro a lista original, e com a ajuda de um apontador auxiliar local, uma segunda lista recebe a primeira, copiando assim os elementos de uma lista para outra, enquanto o comando de repetição for diferente de NULL.

**Mover menor elemento da lista para a primeira posição**

****

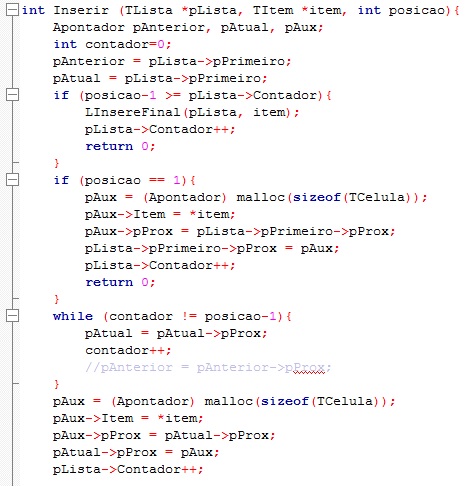
A função percorre toda a lista, então, ela procura o menor elemento da lista, assim que o encontra, move ele para a primeira posição da lista, e o que antes era a primeira posição, passa a ser a segunda, a partir daí, a lista é a mesma com a antiga posição +1.

**Remover elemento específico da lista**

****

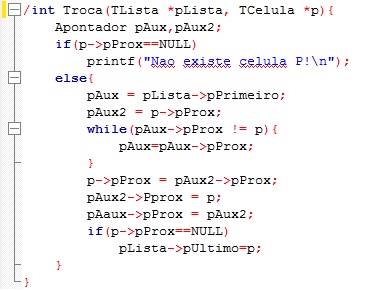
É recebido como parâmetro, uma lisa e um posição. Definindo dois auxiliares locais. Primeiro, é testado se a lista é vazia, depois, se a posição é invalida. Os dois auxiliares recebem o endereço da lista. Então, um comando de repetição foi usado para guardar a posição para onde aponta o anterior a posição desejada. O segundo auxiliar recebe esse endereço de memoria. Então, foi pego a posição para onde o item a ser removido apontava e este passou a ser o novo endereço da posição anterior o item. Por fim, é testado se o item se localiza no final da lista.

**Inserir elemento em posição especifica da lista**

****

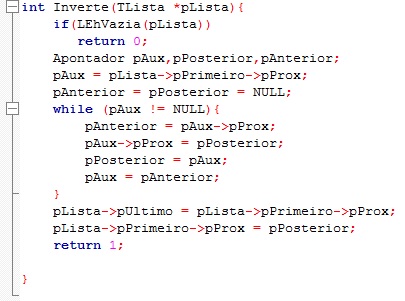
É recebido como parâmetro, uma lisa, uma posição e um item a ser inserido. Definindo três auxiliares locais. Dois desses auxiliares, anterior e posterior, recebem o endereço da lista. Primeiro, é testado se a posição corresponde ao final da lista, caso seja, a operação de inserir ao final é executada. Depois, é testado se esta posição corresponde ao inicio da lista. Caso nenhuma das condições acima sejam verdadeiras, um comando de repetição é executado para um auxiliar receber a posição anterior à posição no qual deseja ser inserido. Então, o próximo item depois da posição desejada tem seu endereço guardado para que o novo item aponte para esse local. Ao final, o contador, usado para saber o tamanho da lita, é incrementado.

**Trocar posições de células**

****

Recebe como parâmetro uma lista e um célula. Com a ajuda de dois apontadores auxiliares locais, é testado se a célula existe, caso contrário, um dos auxiliares aponta para o começo da lista, enquanto o segundo aponta para a célula. Dentro de um comando de repetição, é executado a troca de posições.

**Inverter toda a lista**

****

Recebe como parâmetro a lista a ser invertida. Primeiro a função testa se a lista recebida é vazia, depois, com a ajuda de três apontadores locais auxiliares, o primeiro, recebe a lista, enquanto um comando de repetição é executado diferente de NULL, os outros dois auxiliares vão percorrendo toda a lista usando manipulações entre si, armazenando sempre o ultimo e o próximo, invertendo assim, item por item.

**Dividir em K listas**



Recebe como parâmetro o endereço vetor de listas da função *main*, pois depois será retornado *listas\_atuais,* variável usada para controlar a quantidade de listas criadas; recebe também o endereço da lista a ser dividida, o inteiro *K*, com a quantidade de listas a serem divididas e o inteiro *listas\_atuais*, que controla a quantidade de listas criadas. Verifica se a quantidade de células é divisível pela quantidade *k* de listas: caso seja, todas as novas listas terão o mesmo número de células, caso contrário, serão criadas k-1 listas, com quantidade iguais de células e depois será criada a última lista, que irá conter as restantes células. O retorno *listas\_atuais* será salva na variável de mesmo nome da função principal, que será atualizada.

**Conclusão**

Ao iniciar os estudos em programação, logo vemos que é necessário muita dedicação e mão na massa, para que um dia possamos dizer “hoje eu sei programar da forma correta”. O trabalho prático serviu como um pilar para o caminho nessa jornada. Pois nos permitiu trabalhar o mais perto possível com os ponteiros, essenciais na linguagem C. Algumas funções foram complicadas, mas com o auxilio de desenhos, acabou se tornando mais claro e evidente.

Portanto, vimos que a prática é de extrema importância, principalmente nas disciplinas da Ciência da Computação, pois é assim que conseguimos absorver conhecimento e realmente aprender.