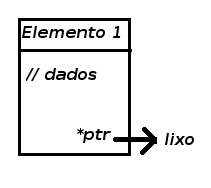
Estrutura de dados dinâmica realmente não é o assunto mais fácil da linguagem C, é realmente necessário estudar bastante para entender perfeitamente os conceitos e ideias por trás deste assunto.  
  
Visando clarear mais a mente de nossos estudantes, a [apostila C Progressivo](http://www.cprogressivo.net/) vai explicar o conceito de lista apenas com figuras e explicações escritas, nada de código por hora.

* [Obtenha sua certificação em C e entre no Mercado de Trabalho!](http://afiliado.bravacursos.com.br/4573/curso-de-programacao-em-c/)
* Leia este tutorial no seu computador ou celular: [Apostila C Progressivo](https://www.cprogressivo.net/p/apostila-c-progressivo.html)

**Lista encadeada - O nó**

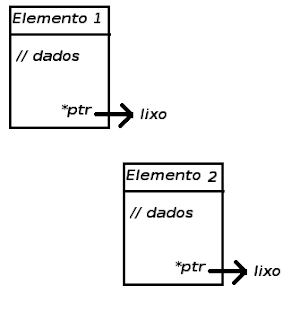
Vamos tentar explicar, sem pressa e com detalhes, cada uma das ideias e elementos necessários para se estudar e trabalhar com estruturas de dados dinâmicas.  
  
O objetivo deste tutorial de nossa apostila é que você entenda, na sua cabeça, como tudo funciona.  
E esta é uma dica valiosa: antes de programar e fazer seu aplicativo rodar, ele tem que rodar perfeitamente na sua cabeça.  Ou seja: antes de começar, entenda e faça na mente.  
  
Vamos chamar cada struct, a partir de agora, de **nó**.  
  
Por exemplo, criamos uma struct com algumas variáveis e um ponteiro que aponta para o seu próprio tipo. Logo, quando criamos tal nó, é da seguinte forma:

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEixm3Cv00CZcn8V3Q8DY4djFegUy9oeYPHWEQE15TvE7A1MnoOU12aa-saZ4ZGHx7lNTSfl6RGGrrCshpTcyvecwuTiXZVv7FBNeH7jontg0Aq1-DQ-kvJ3lFQvBGwVvnuKevHg3mu-UbI/s1600/Listas-em-C-1.png)

Pronto. Esse nó está em algum ponto da memória, e seu ponteiro aponta para um endereço de memória qualquer.

Agora vamos declarar outro nó, o *Elemento 2****.***Seu funcionamento e criação ([alocando memória de maneira dinâmica](http://www.cprogressivo.net/p/alocacao-dinamica-de-memoria-em-c.html)) são idênticos ao *Elemento 1*, e essa estrutura, ou nó, está em um lugar qualquer da memória, bem como seu ponteiro aponta para um local aleatório, chamado de lixo.

Veja:

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEj0xWxlRW0g-StHk60X3ZEeakw5cDGetpYEZDmQUOt_fgAdonxBGyUepmH-fV6CgsYHDkvGhyphenhyphenN9mbxHJ6_0eYSFIJ4bZ4iDGGGd8eldqyzgymPzKehHbFSEX-GeJDRmD8sMM9TPRDrFE78/s1600/Listas-em-C-2.png)

**Conectando os nós de uma lista**

Agora temos que 'ligar', 'conectar' ou 'encadear' esses dois nós.

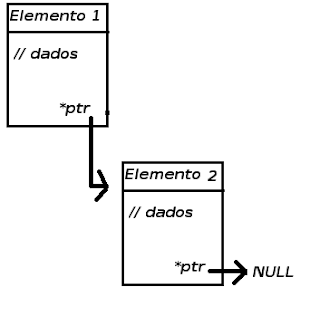
Lembre-se que o ponteiro é para o tipo do nó, então podemos fazer com que o ponteiro do primeiro nó aponte para o segundo nó! E como o nó foi alocado de maneira dinâmica, ele também é representado por um endereço de memória, que será usado pelo ponteiro do elemento 1.

No nó 1, fazemos:

ptr = Elemento2;

Porém, o ponteiro do segundo nó ainda aponta para outro endereço de memória (lixo).

Para podermos identificar que esse elemento é o último da lista, vamos fazer com que o ponteiro do nó 2 aponte para NULL, veja como ficou nossa lista encadeada:

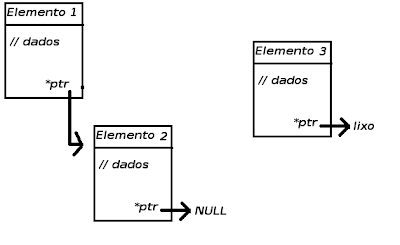
[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEiY0L4CURCaBZM6neRAu8EWLFT79SLkYQ_HMSRyP-f5x9novCfs-B67iGh0R788NmdKMgXFdzkpzozChkx0tPw7nisfd8I5Zi1N6zDI3BrWxiCcx9jYr-M9GvIyFVUUYLh_8qIRTmtc4w4/s1600/Listas-em-C-3.png)

 Pronto! Temos uma lista simples, com duas estruturas conectadas.

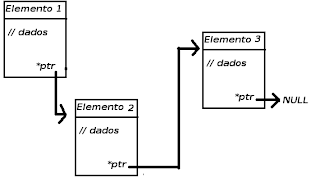
|  |  |
| --- | --- |
|  |  |

**Inserindo nó ao final da lista**

Vamos colocar outro nó na lista?  
  
Primeiro declaramos o elemento 3.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjPErtQyYdodx0T8en3oZsrEyjX71n2rF3NxiMskgwzr5o-7On6oglf60NdRGEJAAsPvQ9oQqzloLIyCcrQU6YCfDzJDyhF4mM6ocP1v7xwQWe0wjMm-_8oEiY8jRpZWU1u_83tIIkNUI8/s1600/Listas-em-C-4.png)

Como queremos que ele fique depois do elemento 2, fazemos com que o ponteiro deste segundo elemento aponte para o nó 3. E para identificar o nó 3 como o último da lista, fazemos seu ponteiro apontar para NULL. Veja:

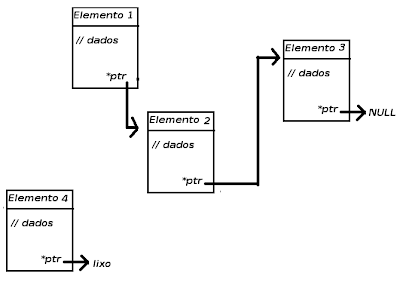
[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjQYETUBemeaRN3_dhGIxxL3Aigs_GrXHx0oU_rlz0lzEofi3_iOY7_19XIsj_un_AvawiOGDxNxPraReoviLkuOg42w7NFu5lX0uzJpAqKFmvv-H3vODzRopI9-1tyLNOKd4GlOo6awPo/s1600/Listas-em-C-5.png)

 Simples, não? Agora raciocine como seria para colocar um quarto nó ao final da lista.  
O que você faria? Quem apontaria para quem? E qual ponteiro vai apontar para NULL agora?

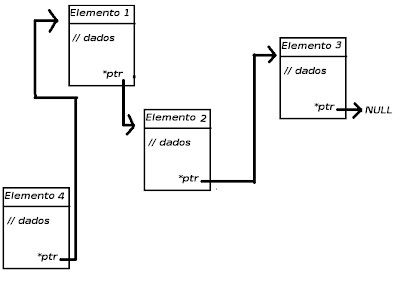
**Conectando nós no início da lista**

Nos exemplos acima, conectamos nós ao final da lista.  
Basicamente fazemos com que o nó antigo, que era o último, aponte para o novo nó, e o ponteiro desse novo nó aponte para NULL: é o nosso algoritmo para inserir um nó ao fim da lista.

Agora iremos aprender como inserir o nó no COMEÇO da lista, na primeira posição!  
Como antes, primeiro declaramos um novo nó, o *Elemento 4*.  
Ele está em algum lugar na memória, perdido e solitário, assim como o seu ponteiro.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEj2PKuFxc24CleqoI-_ZUj8s0JoOeqN452mtdeX6e4IkmrZk67r6-ywTr6S0NTQ8R0uKdASDxyuU1qIIw34Q-w6ebB5fpFA8aVFSXQCMqNRb0iIu0fegsFy3aL8q2vcZnoDWxIqGu17GBQ/s1600/Listas-em-C-6.png)

Precisamos, agora, conectar esse novo nó na lista.  
Porém, ninguém vai apontar para ele desta vez. Agora é ele que vai apontar para o primeiro nó da lista, que é sempre chamado de **head** (ou **cabeça**, em português).  
  
Ou seja, no ponteiro do novo nó, fazemos:  
ptr = Elemento1 ou ptr=cabeca

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEi2-oEWz17wPNMHjF4JZ4BW8ieC83h7p8uH_Gq7k5reqb-lfbDkYDitp2oChnHeyotzvAS_gmksIruvOS9zsklDOnk7emFu1gjVB6AkzZ_8_H_QFbym4WKAAL402yrQ4hDAiCuCcodXkr0/s1600/Listas-em-C-7.png)

E pronto! Adicionamos o nó 4 no início da lista. Simples e óbvio, não?

**Inserindo um nó no meio da lista**

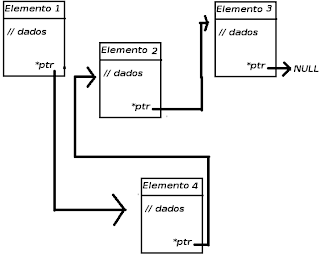
Vimos que para adicionar um nó no início da lista, basta o novo primeiro nó apontar para o antigo primeiro nó. Já para colocar ao fim da lista, simplesmente apontávamos o ponteiro do último nó para o novo nó.

E para adicionar no meio da lista? Fazemos as duas coisas.

Por exemplo, para colocar o nó na segunda posição: primeiro declaramos o nó 4, que está em um lugar qualquer da memória.

Agora fazemos com que o primeiro ponteiro deixe apontar para o nó 2 e aponte para o novo nó, o nó 4.

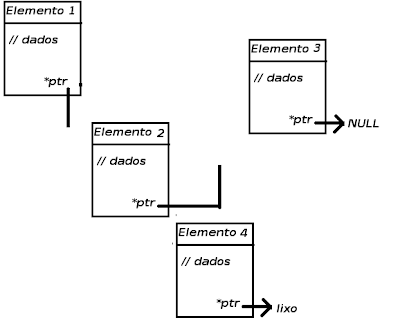
E agora fazemos com que o nó 4 aponte para o nó 2.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEiHUzMvmOgVDD-X2rAZaxg5-bUL9Qa1qAvnpO4iN4tbsU3APdUdQYy2pQL5gVwAHuJGZ260Ftv6nTcWF0daZMP4R2P_W7gaf1NrZs9TRzl_T-JTbMq769YVrWE-Nw_1s3y8XCpOwgdGT40/s1600/Listas-em-C-10.png)

**Excluindo um nó da lista**

Para excluir um nó de uma lista, devemos fazer com que esse nó não aponte para ninguém e que ninguém da lista aponte para ele. Ou seja, ele não pode se relacionar com nenhum nó, deve se isolar.

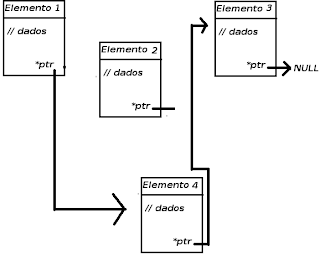
No exemplo anterior, se quisermos tirar o nó 2, é só fazermos os passos contrários.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhpU485nU7sQAaD9IA1L2mSmU3UgpocqB1qidmkvVRwGHG9ge9I9W_DMPK_pD9uGLmEMn26qKzOzofatuLNisDlkWNAQKAa7ksy7fXkFq_gJU1FtCALrfc-B3z8uAFSHO3yYp4OwqGyYjk/s1600/Listas-em-C-8.png)

O nó 4 apontava para o 2. Vamos cortar essa relação: agora o nó 4 não vai mais apontar par ao nó 2, e sim para o nó 3. Podemos ainda fazer com que o nó 2 que apontava para o nó 3, aponte para outro lugar.

Pronto. Ninguém aponta para o nó 2, e ele não aponta para ninguém.

O nó 1 aponta para o 4, que aponta para o 3, que aponta para NULL. Cadê o nó 2? Não faz mais parte, vai ficar lá perdido na memória, pois não vai se relacionar com ninguém da lista.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEi1-zzffO7ICFWSZSz7Xa6L6Xa3KzcfrepbYK9H9p3V7E3fdADUEYft6kPr-3z4CB8Pf1fMsSGc0eOBT4-4eF61_45XVhtw4L7nAo3tnjIsC0RBb5HbFW_kOaEb4prEejkgIFfYQYLvulQ/s1600/Listas-em-C-9.png)

E para excluir o primeiro nó?  
Ora, o nó 1 só faz apontar para o nó 2. Se fizermos ele apontar para NULL, ele sai da lista, já que ele é o primeiro e ninguém aponta para ele.  
  
E para excluir o último nó ?  
Basta fazer com que o penúltimo nó, que apontava para o último, deixe de apontar para ele e aponte para NULL. Pronto, agora ninguém aponta pro último e ele ficou apontando para NULL, logo, não tem mais nenhuma relação com a lista encadeada.  
  
Simples, bonitinho, com desenhos e tudo fazendo sentido.  
Se não entender, leia, releia até entender. Só podemos ir para o próximo passo depois de entendido perfeitamente este tutorial de nosso curso.  
  
Caso tenha entendido, é hora de partir para os códigos em C.  
No próximo tutorial vamos ensinar como programar uma lista encadeada em C. Até lá.