Listas Encadeadas (linked list)

Roberto Araujo
SI – UFPA/2013

Listas Encadeadas

- É uma estrutura de dados que representa uma sequência de NOs (objetos) de dados
- Um nó consiste de **duas partes**: uma área de dados e uma área de ponteiros

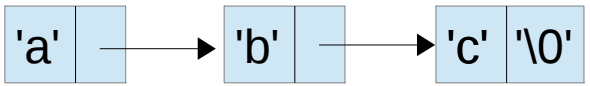
- **Área de dados**

  Pode conter qualquer tipo de dado (int, float, vetor,...)

- **Área de ponteiros**

  Contém no mínimo um ponteiro e aponta para outro nó.

- Diferentemente dos vetores, onde a memória é alocada consecutivamente, cada NÓ pode estar em diferentes partes da memória.
- A **parte de ponteiro** localiza um NÓ, dentro da memória, que faz parte da lista



Exemplo de lista encadeada simples contendo três nós

Definindo o NÓ da Lista

- O NÓ que formará a lista encadeada é definido através de uma estrutura. Ela conterá a parte de dados e a parte de ponteiro.

  Ex:

```
typedef struct testeno {
    char dado;
    struct testeno *ptr;
} NO;
```

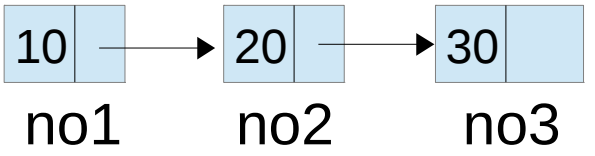
Parte de dados

Parte de ponteiro. Contém um ponteiro para o próximo nó do mesmo tipo

Exemplo de Lista Encadeada Simples

(**SEM** alocação dinâminca de memória)

```
#include <stdio.h>
 main() {
   typedef struct testeno {
      int valor;
      struct testeno *proxno;
    } NO;

    NO no1, no2, no3;
    int i;

    no1.valor = 10;
    no2.valor = 20;
    no3.valor = 30;

    no1.proxno = &no2;
    no2.proxno = &no3;

    i = no1.proxno->valor;

    printf("%i - ", i);
    printf("%i\n", no2.proxno->valor);
}
```



Exercício

- Escreva um programa para armazenar a idade e o salário de cinco indivíduos em uma lista encadeada simples, sem alocação dinâmica de memória.

Alocação Dinâmica de Memória

- Diferente dos vetores que possuem um tamanho fixo e são definidos em tempo de compilação, a lista encadeada pode crescer, ou seja, novos nós podem ser adicionados a lista.

- A cada novo NÓ, precisamos alocar memória para armazená-lo e ligá-lo a lista corrente. Tal alocação é realizada em tempo de execução.

- Dessa forma, os NÓs são criados dinamicamente, i.e. de acordo com a necessidade e enquanto exista memória disponível.

- Através da alocação dinâmica podemos manter na memória somente o necessário, sem desperdiçar espaço.

Alocando Memória Dinamicamente em C

- Para alocar memória dinamicamente, utilizaremos a função **malloc()** contida na biblioteca **<stdlib.h>**
- Esta função recebe como argumento o número de bytes requeridos e retorna um ponteiro (endereço) para o novo bloco de bytes.
- **malloc()** retorna um ponteiro NULL caso não a memória requerida não estejaa disponível
- Para definir o número de bytes necessário a função malloc(), utilizaremos a função **sizeof()**
- Se precisamos alocar memória para um nó, precisamos passar o tamanho da estrutura para o malloc() e definir um ponteiro para estrutura para retorno do malloc().

Ex:

```
NO *ptr;
ptr = malloc( sizeof(NO) );
```

Lista Encadeada
Operações Comuns

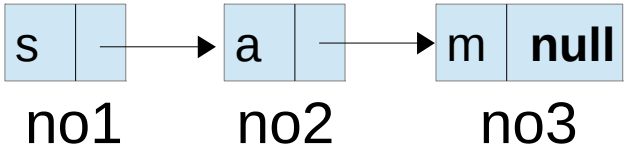
- Adicionar NÓ na lista
  - Início
  - Meio
  - Fim
- Apresentar a Lista
- Buscar elementos na lista
- Remover NÓ da lista
  - Início
  - Meio
  - Fim

Inicializando uma Lista Encadeada Simples

(COM alocação dinâmica de memória)

```
#include <stdio.h>
#include <stdlib.h>
   typedef struct no {
       char dado;
       struct testeno *proxno;
    } LNO;

 main() {

   LNO *no1=null, *no2=null,
           *no3=null;

   no1 = malloc(sizeof(LNO));
   no2 = malloc(sizeof(LNO));
   no3 = malloc(sizeof(LNO));

    no1->dado = 's';
    no1->proxno = no2;

   no2->dado = 'a';
   no2->proxno = no3;

   no3->dado = 'm';
   no3->proxno = NULL;
}
```

Definindo o NÓ



Alocando espaço para os 'NOs'

Atribuindo valores para um NÓ

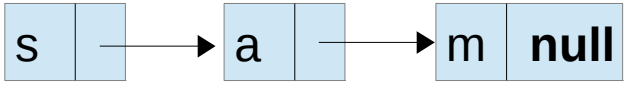
O último NÓ da lista não aponta para nenhum NÓ

Apresentando os Dados
Lista Encadeada Simples

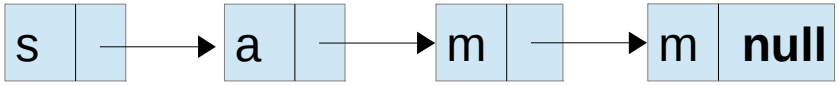
```
void mostrar_lista ( LNO *p ) {

    while ( p != NULL ) {
        printf("O dado é: %c \n", p->dado);
        p = p->proxno;
    }
}
```

mostrar_lista(p1); ⟶ Ponteiro do primeiro nó

Adicionando um novo NÓ no FINAL
Lista Encadeada



no1 no2 no3



no1 no2 no3 no4

```
void adiciona_no(LNO **p, char letra) {


  LNO *p1 = NULL, *p2 = NULL;
  p1 = *p;
 // Não existe nenhum NÓ na lista
 if (p1 == NULL) {
    p1 = malloc(sizeof(LNO));
    if (p1 != NULL) {
        p1->dado =  letra;
        p1->proxno = NULL;
        *p = p1;
    }
```

```
// Já existem NOs na lista
} else {
  while (p1->proxno != NULL)
        p1 = p1->proxno;
  p2 = malloc(sizeof(LNO));
  If (p2 != NULL) {
            p2->dado = letra;
            p2->proxno = NULL;
            p1->proxno = p2;
   }
  }
}
```

Obs.: adiciona_no(&ptr, 'a')

Adicionando um novo NÓ no FINAL
Lista Encadeada

```
main () {
    LNO *n = NULL;
    char letra;

    do {
        printf("Inf. uma letra");
        letra = getchar();
        if (letra != 'x') {
            adiciona_no(&n, letra);
        }
    } while (letra != 'x');
}
```

IMPORTANTE:

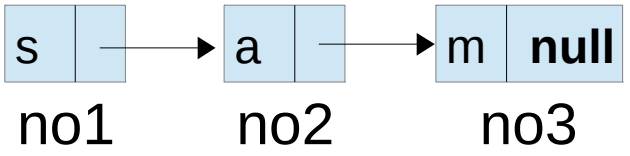
Esse ponteiro deve ser iniciado com **NULL**

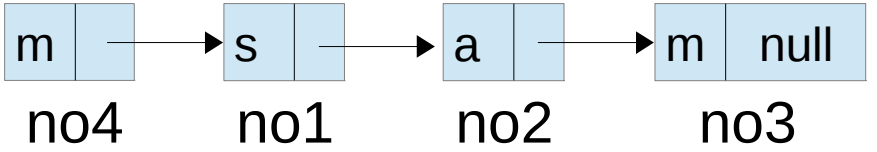
Adicionando um novo NÓ no INÍCIO

Lista Encadeada

Como adicionar um novo NÓ no INÍCIO da lista ?

Adicionando um novo NÓ no INÍCIO
Lista Encadeada






```
void adiciona_ini(LNO **p, char letra) {


  LNO *p1 = NULL;
  p1 = malloc(sizeof(LNO))
  if (p1 != NULL) {
     p1 = malloc(sizeof(LNO));
     if (p1 != NULL) {
         p1->dado =  letra;
         p1->proxno = *p;
         *p = p1;
     }
}
```

Obs.: adiciona_ini(&ptr, 'a')

Adicionando um novo NÓ no MEIO

Lista Encadeada

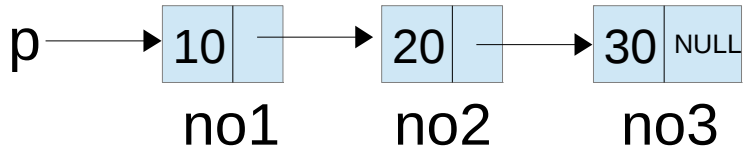
Como adicionar um novo NÓ no MEIO da lista ?

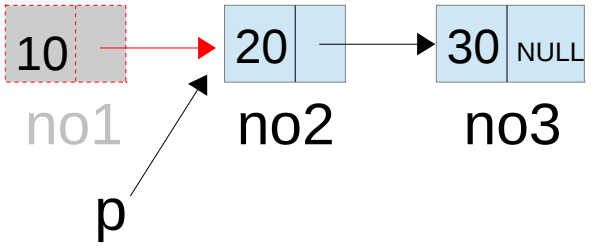
REMOVENDO um NÓ no INÍCIO
Lista Encadeada

- Para liberar espaço de memória utilizaremos a função **free()** presente na biblioteca **stdlib.h**

Ex:

```
void removeno( LNO **p ) {
    LNO *p1 = NULL;
    p1 = *p;
    if ( p1 != NULL ) {
        p1 = p1->prox;
        free(*p);
    }
    *p = p1;
}
```





REMOVENDO um NÓ

Lista Encadeada

Como remover um NÓ no FIM ?

Como remover um NÓ no MEIO?

Busca na Lista
Lista Encadeada

Escreva um função para realizar a procura de um item na lista

Ex:

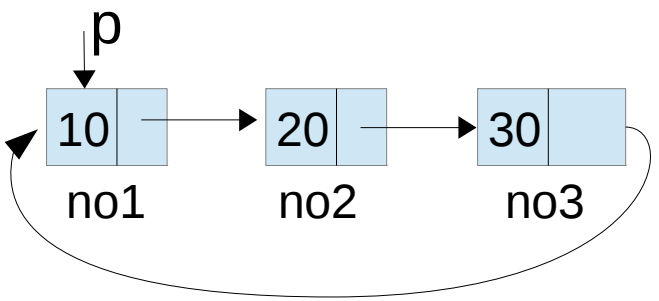
```
int procurar ( LNO *p, int x ) {
    if ( p == NULL)
          return(0);
   } else {
      do {
          if (p->dado == x)
             return(1);
          p = p->link;
       } while ( p != NULL );
      return(0);
} }
```

Outras Listas

- **Lista Encadeada Circular**

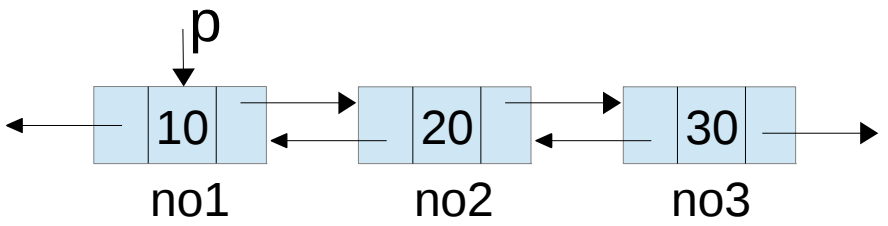
  O último NÓ da lista aponta para o 1o. NÓ

  O primeiro elemento pode ser qualquer elemento da lista



- **Lista Duplamente Encadeada**

  Cada nó tem um ponteiro para o nó anterior e para o próximo nó



Exercícios

1. Desenvolva uma função para retornar o número de NOs de uma lista encadeada

2. Sejam duas listas encadeadas, desenvolva uma função que recebe duas listas encadeadas e retorne a lista encadeada resultante da concatenação

3. Altere o código de uma lista encadeada simples para funcionar como uma lista encadeada circular

4. Altere o código de uma lista encadeada simples para funcionar como uma lista duplamente encadeanda