Listas - Outras

Listas Circulares

Nós Cabeça

Listas Duplamente Ligadas/Encadeadas

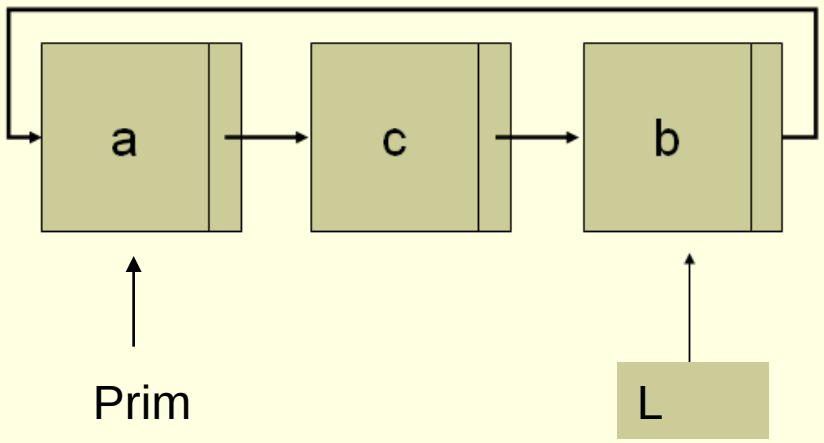
Aplicações

5 e 7/10/2010

Listas Circulares Encadeadas Dinâmicas

- Se o nó next do último nó apontar para o primeiro, teremos uma lista circular.
- Em listas circulares não temos primeiro nem último naturais.
- O esquema abaixo é o mais usual: como não há primeiro nem último nó, fazemos o nó L ser o último e o próximo ser o primeiro.

- Lista vazia: NULL



Representação/Implementação

```
// Declaração da Lista em Lista.c
struct rec {
      elem info;
      struct rec *lig;
};
// Tipo exportado em Lista.h
typedef struct rec *recptr;

// Declaração de uma lista em main.c
recptr  L;
```

Operações

```
void Criar(recptr *L){
     *L= null;
 }
void Insere_Prim(recptr *L, int valor){
recptr p =(recptr) malloc(sizeof(struct rec));
p->info= valor;
p->lig= p;
*L= p;
}
```

Deixem para relatar os erros de Memória insuficiente quando estiverem fazendo o TAD, assim uniformizam todos os retornos de erros.

Insere depois de um nó k

```
void Insere_Depois(elem v, recptr k) {

recptr j =(recptr) malloc(sizeof(struct rec));
j->info = v;
j->lig = k->lig;
k->lig = j;

}
```

Deixem para relatar os erros de Ponteiro Nulo quando estiverem fazendo o TAD, assim uniformizam todos os retornos de erros.

Se a inserção se dá após o último nó (após L) o nó L deve ser modificado

(Façam)

```
void Insere_Depois(recptr *L, elem v, recptr k){

recptr j =(recptr) malloc(sizeof(struct rec));
j->info = v;
j->lig = k->lig;
k->lig = j;
if (*L == k) *L = j;

}
```

Deleta depois de um nó p

- Se houver 1 único nó nós vamos querer deletar ele?
  - Conceitualmente não!
- Se é vazia não podemos deletar nada.

- Mas podemos querer remover mesmo que haja só 1 nó
  - O nó L deve ser modificado.
- Vamos pensar em erros nesta operação,
  - pois não vou fazer a função deleta.

Primeira opção

```
int deleta_depois(recptr p, elem *px){
recptr q;

if ((p == NULL) || (p == p->lig)){
        return 0;
else {
      q=p->lig;
      *px = q->info;
      p->lig = q->lig;
      free(q);
      return 1;
}
}
```

Não pode ser usado para deletar o último nó (nó L).

Segunda opção – deleta mesmo que tenha 1 único nó

```
int deleta_depois(recptr *L, recptr p, elem *px){
recptr q;

if (p ! = NULL) {

   if (p == p->lig)){
            free(p); *L = NULL; return 1;
   }
   else {
             q=p->lig;
             *px = q->info;
             p->lig = q->lig;
             if (q = *L) *L = p; //atualiza nó L
             free(q);
             return 1;
       }
else return 0;
}
```

Vantagens de Listas Circulares

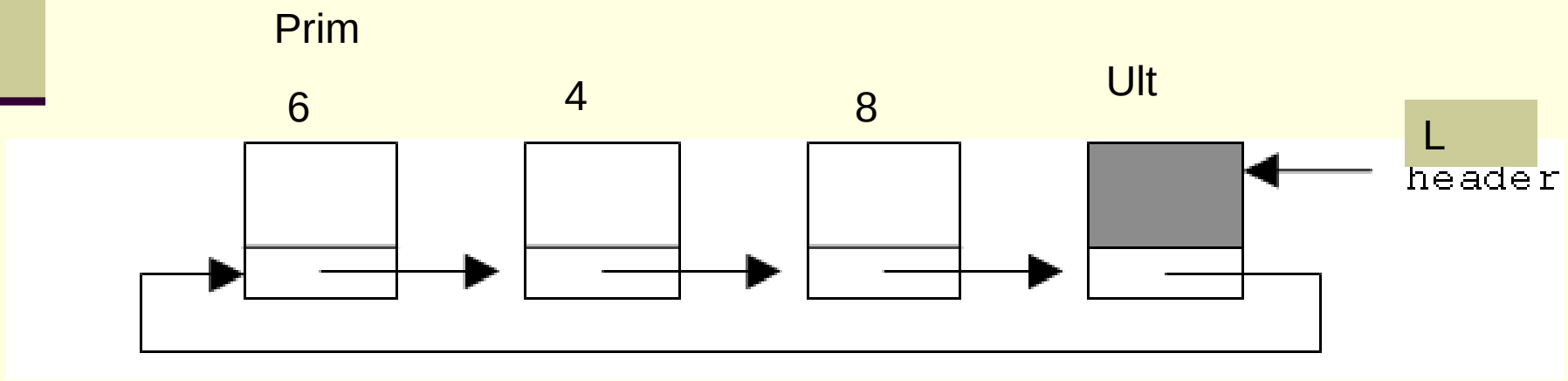
- Podemos varrer a lista completamente,
  - a partir de qualquer ponto e voltar ao início.
- Algumas operações como concatenação e divisão
  - são mais eficazes.
- Exercício: implementem concatenação em Listas Circulares (L1 e L2) passadas como parâmetro, devolvendo o resultado em L1.

Não precisamos percorrer as Listas!

```
void concat(recptr *L1,recptr L2) {
//devolve o resultado em L1
recptr p;
If (*L2 == NULL) return;
If (*L1 == NULL) {
     *L1 = *L2;
     return;
}
p=(*L1)->lig;
(*L1)->lig = (*L2)->lig; // liga L1 ao prim de L2
(*L2)->lig = p; // liga L2 ao prim de L1
*L1 = *L2; // atualiza nova Lista
return;
}
```

Listas Circulares com Nó Cabeça

- Uma lista circular pode ser implementada com um nó “header”, como no diagrama abaixo.
- O esquema abaixo é o mais usual: como não há primeiro nem último nó, fazemos o nó cabeça ser o último e o próximo ser o primeiro.
- Como é uma lista vazia?

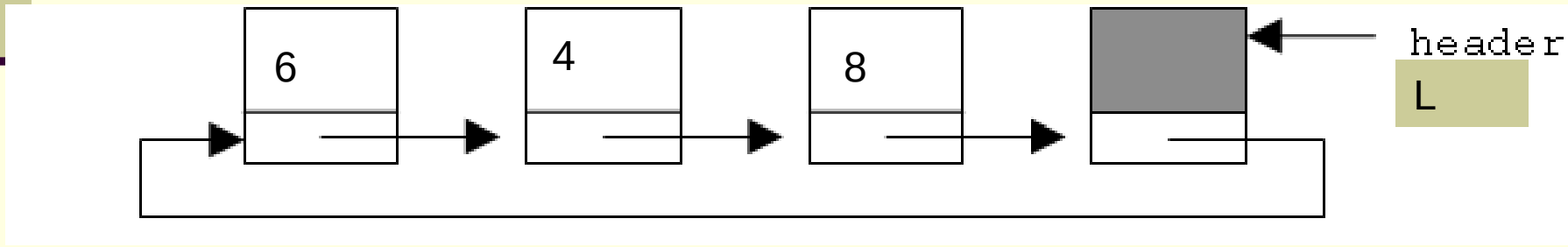


Vantagens

1. Uma das vantagens do “header” é agilizar a busca de um elemento.

   Por exemplo, para achar um elemento X em uma lista NÃO ORDENADA é preciso percorrê-la fazendo duas perguntas:

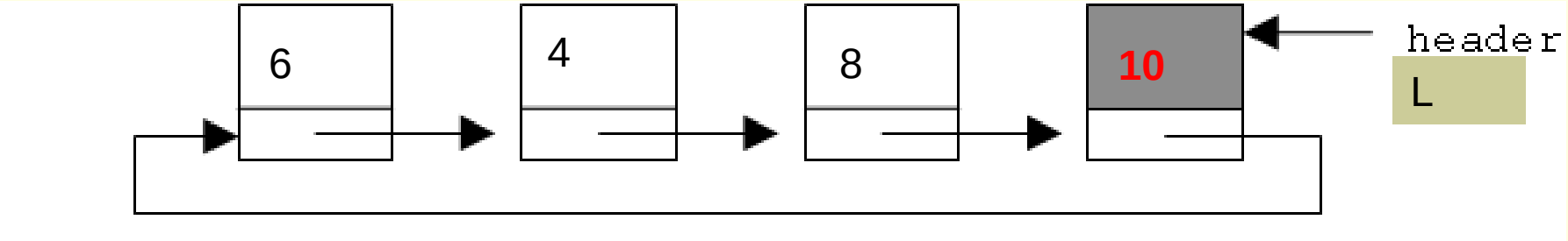
   - (1ª) “achou o elemento?” e
   - (2ª) “já acabou a lista?”



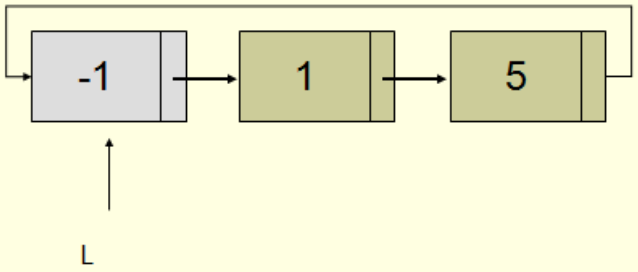
Busca em Lista Simplesmente Encadeada

```
atual= L;    //inicializa "atual" com o início de L
while (atual != NULL) && (x != atual->info)){
     atual= atual->lig; //continua procurando
};  //fim do while
```

Se o valor de X=10 (por exemplo) for atribuído ao “header” antes da busca, o valor de X sempre será encontrado, e a pergunta “já acabou a lista?” não é mais necessária.

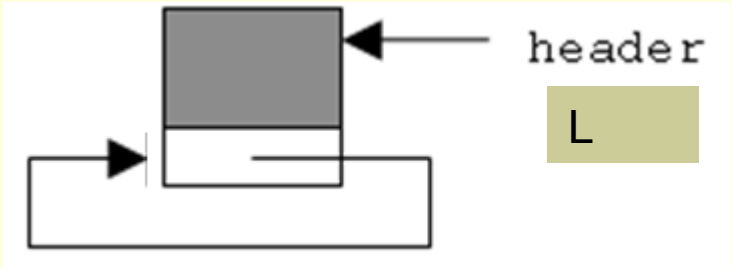


Vantagens



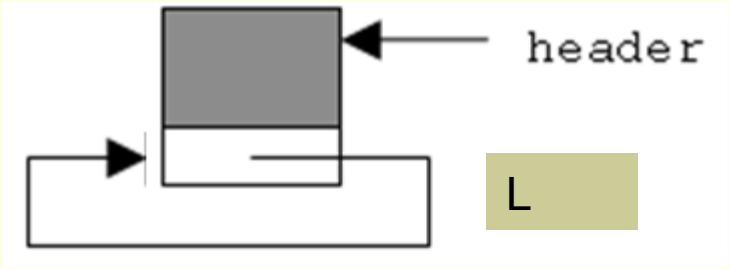
2. Pode-se encontrar a posição do ponteiro Lista após travessia com o ponteiro L.
   - Se colocarmos no nó cabeça um valor especial, diferente dos valores dos nós da lista, este indicará o momento de parar uma busca/travessia na lista.
   - Vamos usar esta vantagem numa aplicação de listas.
3. O ponteiro da Lista (L) nunca muda,

   e a inserção do primeiro elemento não precisa ser tratada como um caso especial!



TAD Lista como Lista Circular com Header

- Implementem uma lista ordenada circular com header,
  - com base no TAD Lista Ordenada simplesmente encadeada já disponibilizado
- Não se esqueçam de que a lista vazia agora não aponta para NULL.
  - Uma lista vazia é aquela em que o header é o único elemento.



Aplicação 1: Inteiros Longos

- Problema: soma de números muito grandes
  - Em Pascal, o tipo integer pode ser de -32.768 a 32.767 ou mesmo ter 32 bits variando de 2147483648..2147483647
  - Longint tem 64 bits, podendo variar de 9223372036854775808 .. 9223372036854775807
  - Variação parecida ocorre com C
  - Como somar números ainda maiores?
    - Listas!

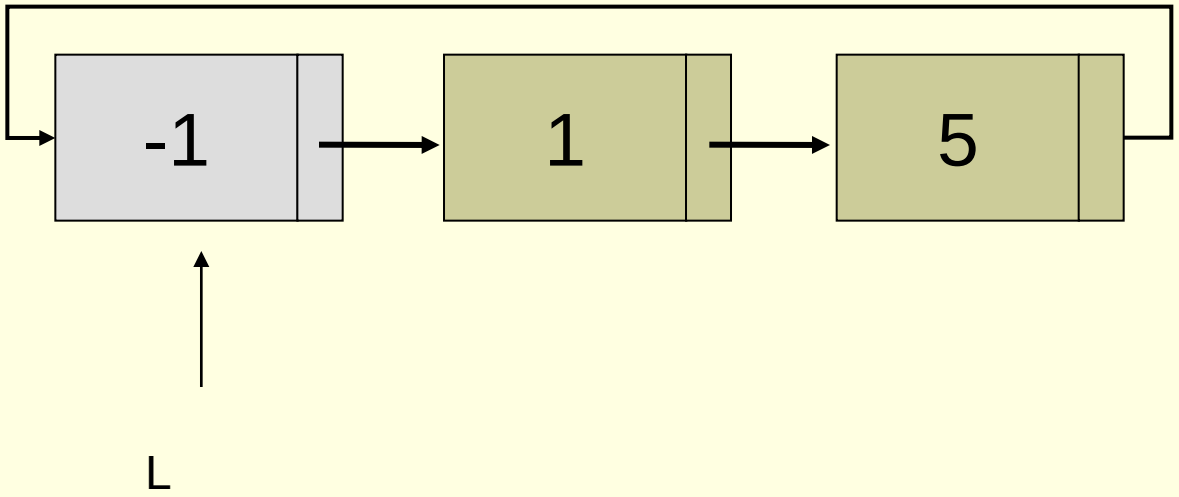
- Adição de inteiros positivos longos com listas circulares e nó cabeça.

```
recptr addint(recptr p, recptr q)
```

- **Qual a vantagem desta estrutura para a aplicação acima?**
  - Travessia com restauração do ponteiro da Lista;
  - Inserção do primeiro nó é igual a qualquer outro

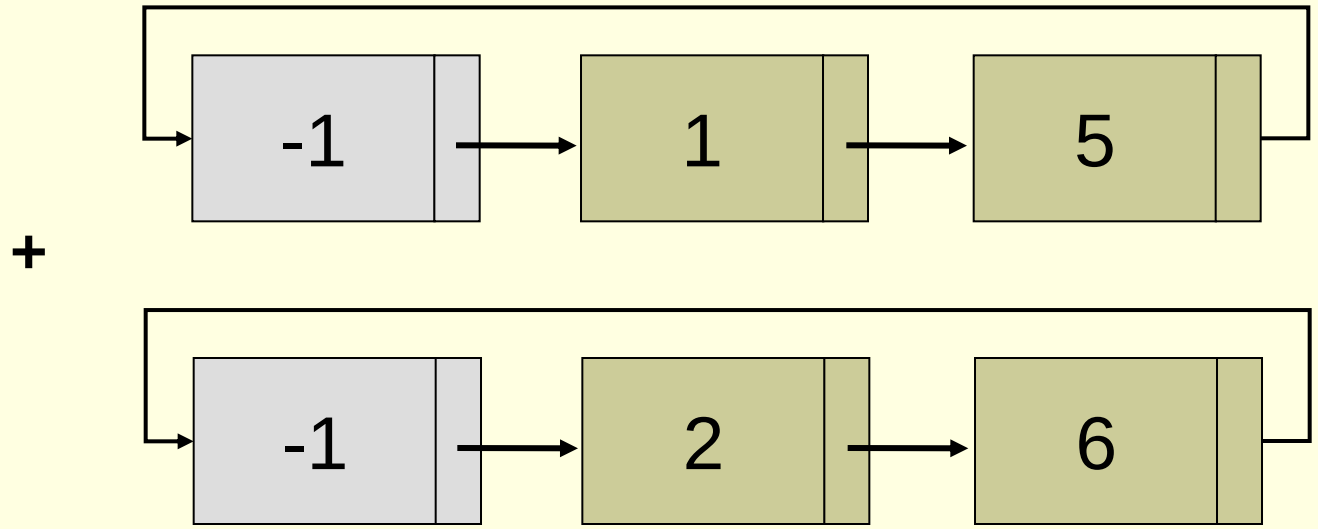
Lista com nó extra; nó cabeça no início

- Representando números como listas
- Nó cabeça tem um valor fora do escopo do problema
  - 15



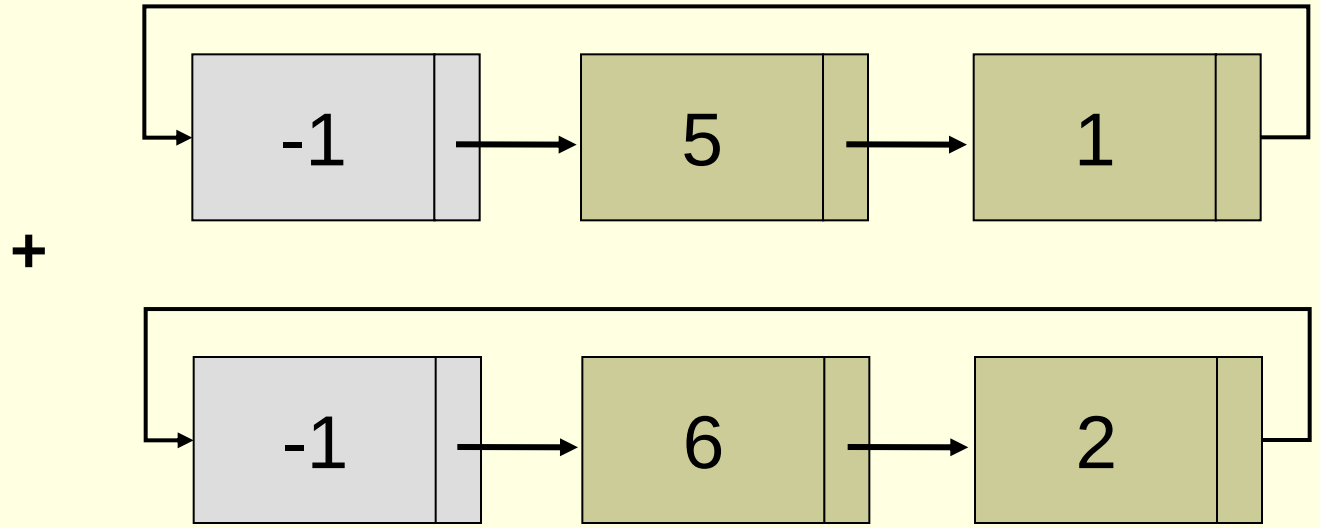
Lista com nó extra

- Para somar dois números representados em listas, somam-se blocos de memória dois a dois, da direita para a esquerda
  - Exemplo: 15+26. Hum.....dá para facilitar?



Lista com nó extra

- Para facilitar nossa vida, números já são representados ao contrário, pois a inserção foi feita sempre na frente
  - Exemplo: 15+26

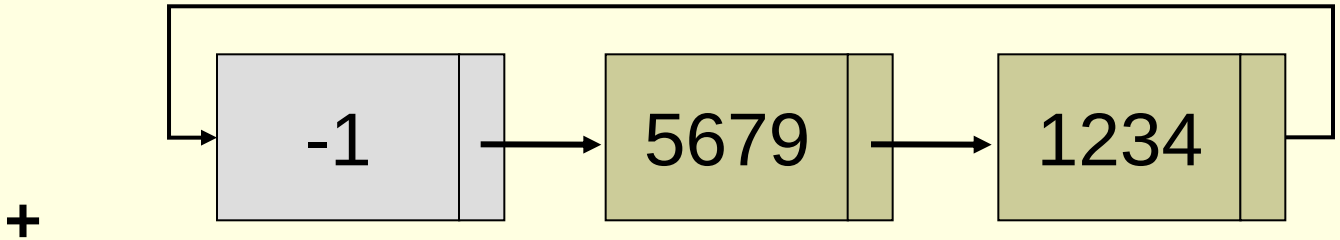


Lista com nó extra

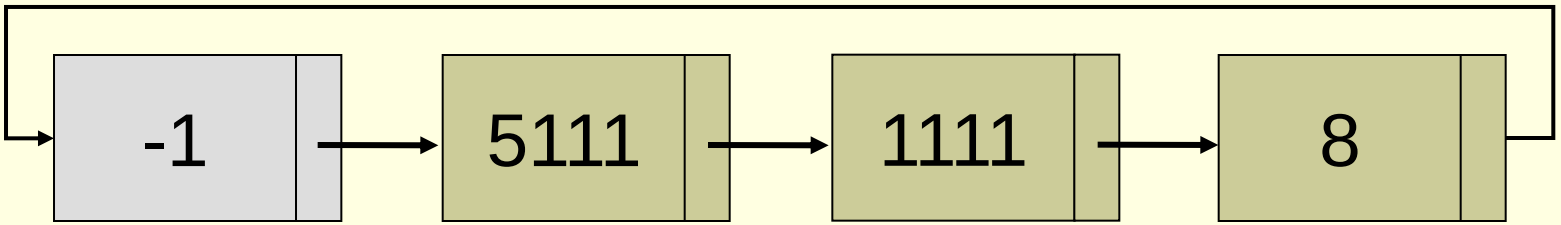
- Exercício: manualmente, represente graficamente e some os números 320.000 e 10.000

Lista com nó extra

- Como os números tratados por esse mecanismo são muito grandes, pode-se aproveitar melhor o tipo inteiro: uso otimizado de memória. Adicionamos 4 digitos a cada vez
  - Exemplo: 12.345.679 + 811.115.111
    - Produza uma outra lista como resultado

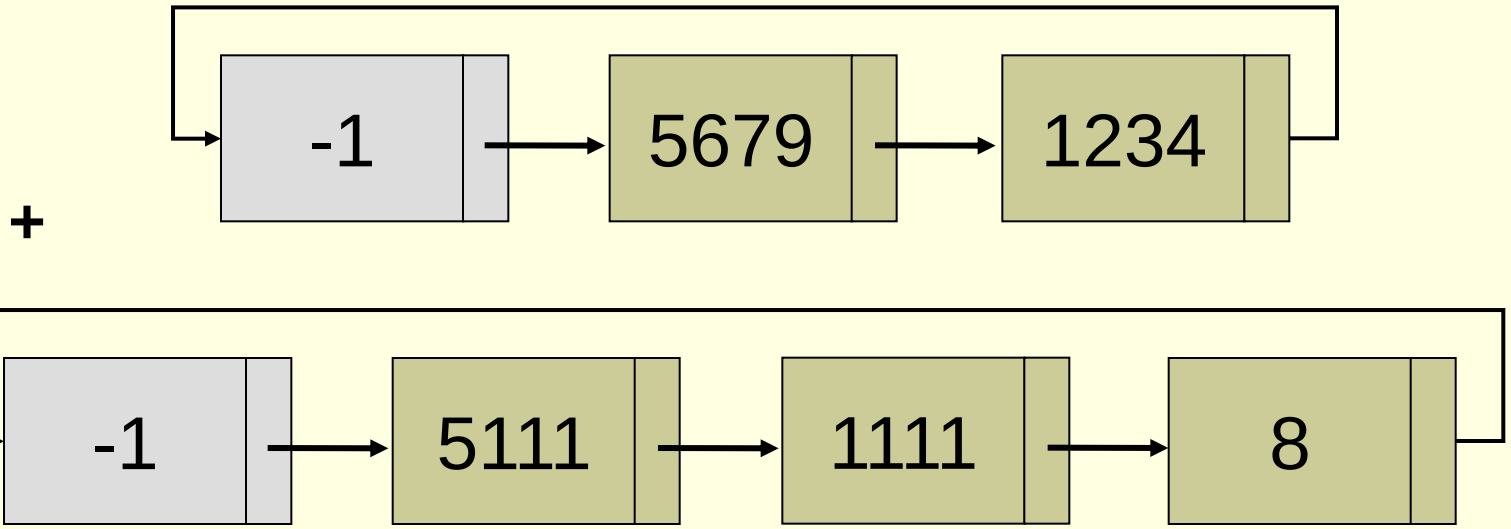


+



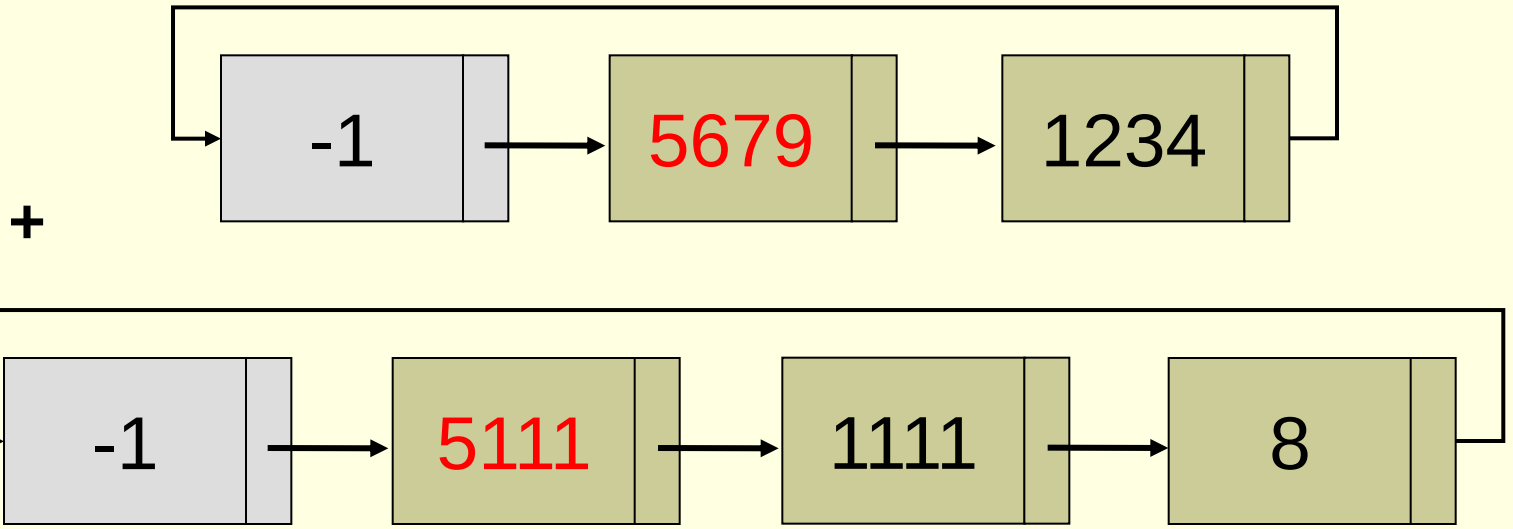
Lista com nó extra

- Como recuperar o número somado para colocar na nova lista?
- Como recuperar o “vai 1”?



Lista com nó extra

- Como recuperar o número somado para colocar na nova lista?
  - Soma mod 10.000 (por que 4 zeros?)
- Como recuperar o “sobe 1”?
  - Soma div 10.000



Lista com nó extra: exercício

- Implementem uma sub-rotina (algoritmo) para somar dois números longos positivos (maiores do que integer) utilizando uma lista circular com nó sentinela/header.
  - As duas listas a serem somadas devem ser passadas por parâmetros, sendo que o ponteiro para a nova lista contendo a soma deve ser retornado em um outro ponteiro, no retorno da função:

```
recptr addint(recptr L1, recptr L2)
```

Algoritmo

- Percorrer as 2 listas enquanto há nós nas 2
  - Somar os 2 nós e mais o carry que inicia em zero
  - Obter o número via soma mod 10.000 e o carry via soma div 10.000
  - Inserir número na nova lista
  - Avançar ponteiros das 3 listas
- Percorrer a Lista 1 se há elementos
- Percorrer a Lista 2 se há elementos
- Se há carry ainda coloque na lista resultante

Implementação com 5 dígitos em cada nó

```
NODEPTR addint(p, q)
NODEPTR p, q;
{
    long int hunthou = 100000L;
    long int carry, number, total;
    NODEPTR s;
    /*  define p e q c/ os nohs posteriores aos cabecalhos */
    p = p->next;
    q = q->next;
    /* configura um noh de cabecalho para a soma */
    s = getnode();
    s->info = -1;
    s->next = s;
    /* inicialmente nao ha transporte */
    carry = 0;
    while (p->info != -1 && q->info != -1) {
        /* soma o info dos dois nohs */
        /* e do transporte anterior  */
```

Continua...a partir do while

```
while (p->info != -1 && q->info != -1) {
    /* soma o info dos dois nohs */
    /* e do transporte anterior  */
    total = p->info + q->info + carry;
    /* Determina os cinco digitos de menor ordem */
    /*              da soma e insere na lista.     */
    number = total % hunthou;
    insafter(s, number);
    /*    avanca os percursos                      */
    s = s->next;
    p = p->next;
    q = q->next;
    /* verifica se existe um transporte */
    carry = total / huntout;
}   /* fim while */
/* neste ponto, devem existir nohs em uma das */
/*              duas listas de entrada        */
```

```
/* neste ponto, devem existir nohs em uma das */
/*                duas listas de entrada          */
while (p->info != -1)    {
    total = p->info + carry;
    number = total % hunthou;
    insafter(s, number);
    carry = total / hunthou;
    s = s->next;
    p = p->next;
}   /* fim while */
while (q->info != -1)   {
    total = q->info + carry;
    number = total % hunthou;
    insafter(s, number);
```

```
    carry = total / hunthou;
    s = s->next;
    q = q->next;
}   /* fim while */
/*   verifica se existe um transp extra a partir dos */
/*                cinco primeiros digitos             */
if (carry == 1)  {
    insafter(s, carry);
    s = s->next;
}   /* fim if */
/*   s aponta p/ o ult noh na soma. s->next    */
/*   aponta para o cabecalho da lista da soma.*/
return(s->next);
/* fim addint */
```

Aplicação 2: somar inteiros longos quaisquer (pos e neg)

- Podemos colocar a informação de sinal no nó cabeça ao invés de usar o nó cabeça simplesmente como sentinela.
  - Para somar um nro pos com um nro neg, o menor valor absoluto deve ser subtraído do maior e o resultado recebe o sinal do maior número em módulo
    - Temos que testar qual é o maior
  - Poderíamos percorrer a lista para responder isto? Maior nro de nós maior número?
    - Prós??? Contras ???

Melhor não percorrer a lista ...caro

- Solução melhor: guardar o tamanho da lista no nó cabeça.
  - Mas o nó cabeça já não guarda o sinal???

- Podemos combinar sinal com tamanho

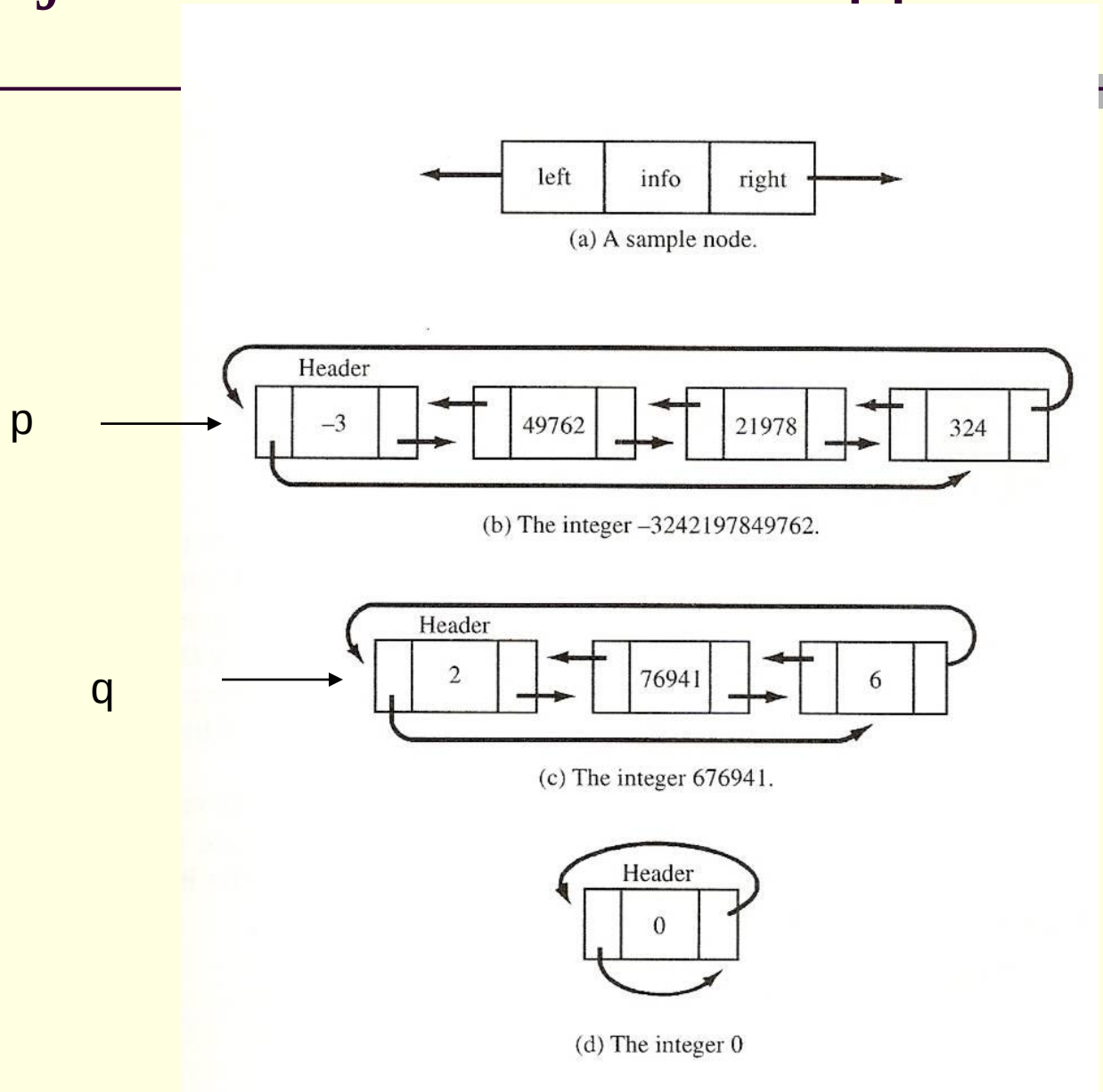
- Mas esta solução não permite usar o nó cabeça para indicar o início da lista....
  - Podemos usar um ponteiro externo para o início da Lista para ser usado para comparação.
- MAS...e se os números de nós dos 2 int forem iguais??? Qual nro é maior?? Como saber???

Nesta implementação, temos que comparar os números, nó a nó do maior dígito para o menor

- Vejam os números:
  - 103.555.888
  - 104.555.789 (maior)

- Estariam nas listas (3 digitos) como:
  - 888.555.103
  - 789.555.104

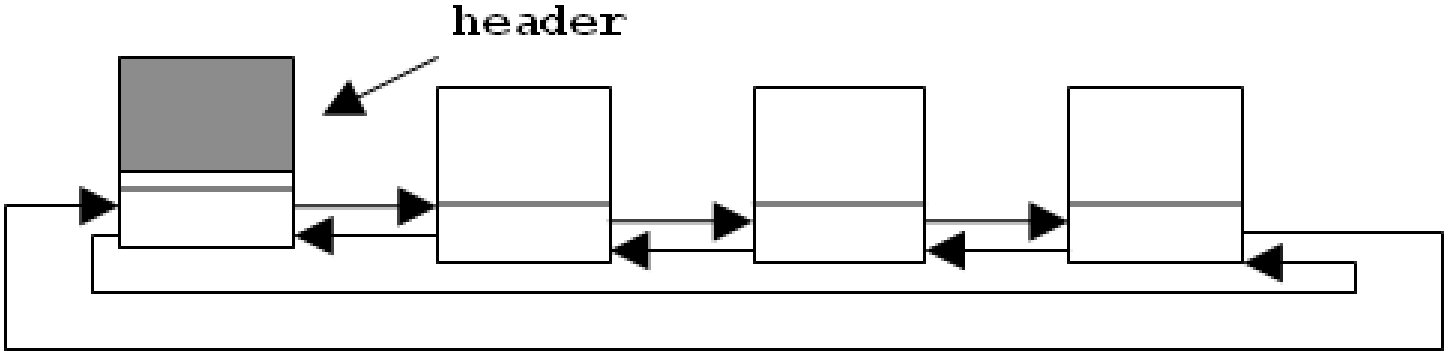
Lista circular duplamente encadeada: ajuda a caminhar da esq para dir



A implementação exige lista duplamente encadeada, pois hora precisamos caminhar à esq e ora à dir

Listas Duplamente ligadas (LDL)

- Em uma lista LDL um elemento aponta para o seu próximo e seu anterior.
- Ela pode ser linear, circular, e pode ou não ter nó cabeça.
- Por exemplo, uma lista circular duplamente ligada (LCDL) com header pode ser esquematizada como na figura abaixo.



Representação/Implementação

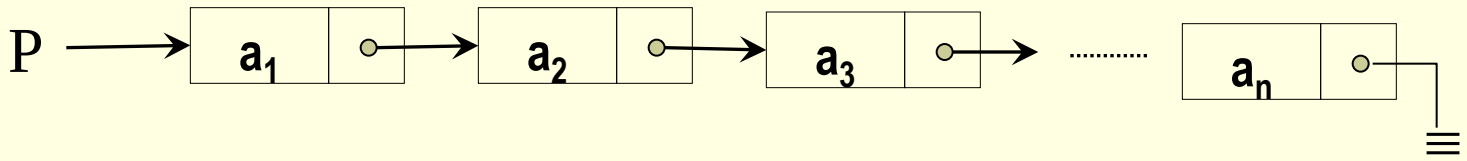
```
struct rec {
      int info;
      struct rec *esq, *dir;
};
typedef struct rec *recptr;
recptr  L;
```

Encadeada/Dinâmica

Aplicação 3: Listas Ativas

- Uma lista (a_1, a_2, ..., a_n) pode ser definida como uma seqüência constituída do elemento a_1 seguido da lista (a_2, ..., a_n) que é definida recursivamente, de forma análoga, até que a lista (a_n) seja formada por a_n seguido da lista vazia ().

Implementação dinâmica reflete essa situação:

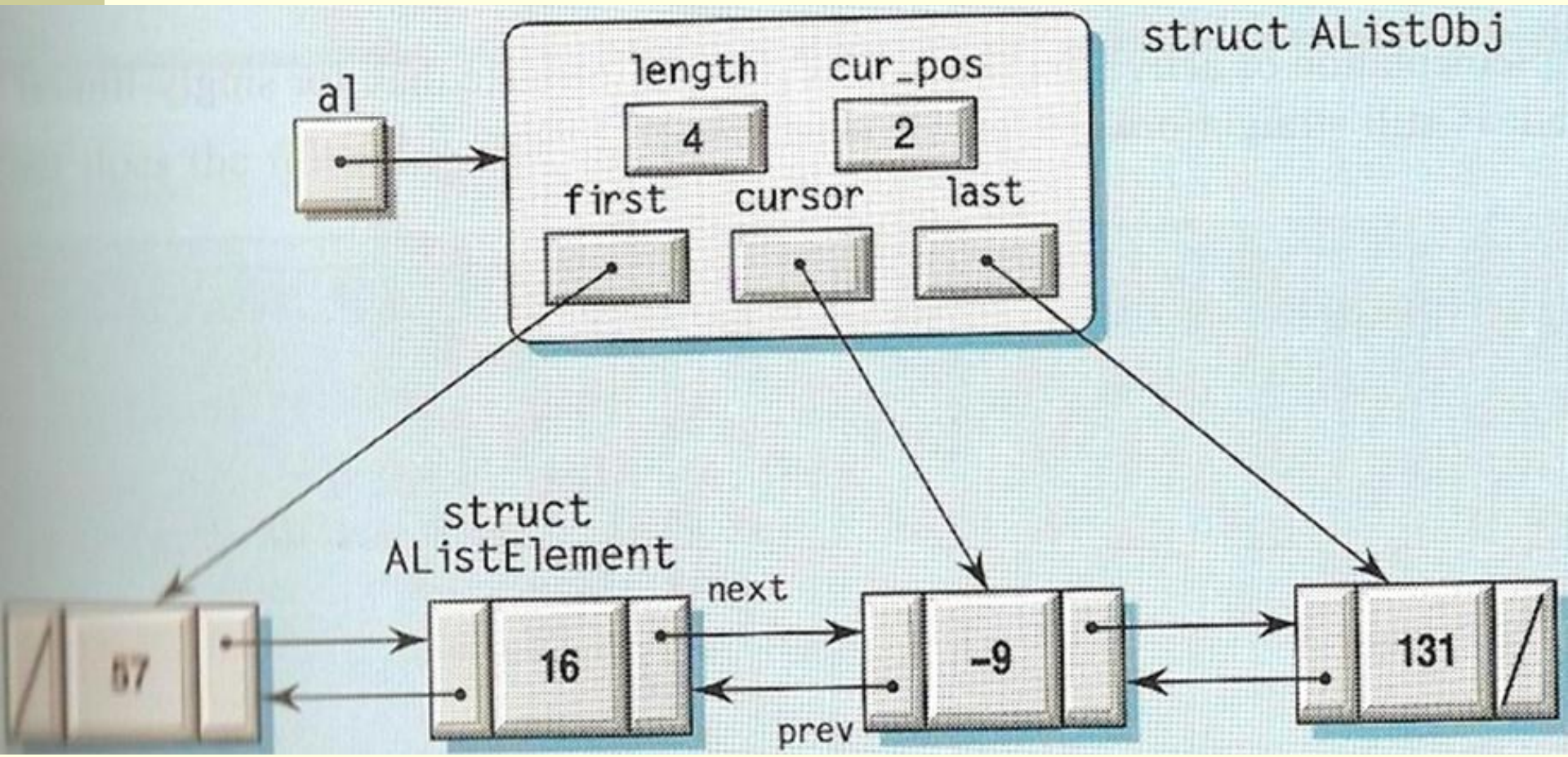


(P e o elemento apontado por P^.lig são do mesmo tipo)

Para remover e inserir precisamos percorrer a lista...

- Para achar o ponto de remoção e de inserção.
- E se víssemos a lista de outra forma: com a possibilidade do programa cliente nomear dinamicamente um particular elemento sobre o qual são operados inserção, remoção, impressão, etc.?
  - Uma lista com um cursor e operações para movê-lo na lista é chamada de Lista Ativa.
- A operação SetaCursor prepara a lista para as próximas operações

Representação de uma Lista Ativa



0 1 2 3

Definição de uma Lista Ativa

```
#define  AListObj                _concat(AList, Obj)
#define  AListElement            _concat(AList, Element)
typedef struct AListObj          *AList;
typedef struct AListElement      *AListLink;
struct AListElement {
    AListLink  next, prev;                  /* Linkage fields */
    AListItem  item;                        /* Data stored */
};
struct AListObj {
    AListLink  cursor;                      /* Current element ref. */
    AListLink  first, last;                 /* Ends of the sequence */
    size_t     length;                      /* Number of items stored */
    size_t     cur_pos;                     /* Position of cursor (origin 0) */
};
```

TAD da Lista Ativa

construtores

inspectores

modificadores

```
AList AList_Create(void);                          Post: IsEmpty(return_value)
void AList_InsertBefore(AList al,
                        AListItem value);
void AList_InsertAfter(AList al,
                       AListItem value);
Bool AList_IsEmpty(AList al);                      Post: return_value = len(al) = 0
size_t AList_Length(AList al);                     Post: return_value = len(al)
size_t AList_GetCursor (AList al);                 Post: return_value = cur(al)
Bool AList_CursorDefined(AList al);                Post: return_value = cur(al) <len(al)
AListItem AList_Inspect(AList al);                 Pre: cur(al) < len(al)
void AList_SetCursor(AList al,                     Post: cur(al') = min(pos, len(al))
                     size_t pos);
void AList_Advance(AList al);                      Post: cur(al') = (cur(al) + 1) % (len(al)+1)
void AList_Backup(AList al);                       Post: cur(al') = (cur(al) - 1) % (len(al)+1)
AListItem AList_Replace(AList al,
                        AListItem value);          Pre: cur(al) < len(al)
AListItem AList_Delete(AList al);                  Pre: cur(al) < len(al)
```

Vejam que a implementação é duplamente encadeada

- O controle da seqüência de elementos é do cliente
- O controle da implementação em memória é do TAD

- SetCursor é de custo constante: O(1)
- Precisamos que seja duplamente encadeada para que
  - as operações a qualquer elemento interior (Advance e Backup) sejam feitas em tempo constante

Aplicação 4: Manipulação de polinômios

$p(x) = a_0x^0 + a_1x^1 + a_2x^2 + a_3x^3 + ... + a_nx^n$

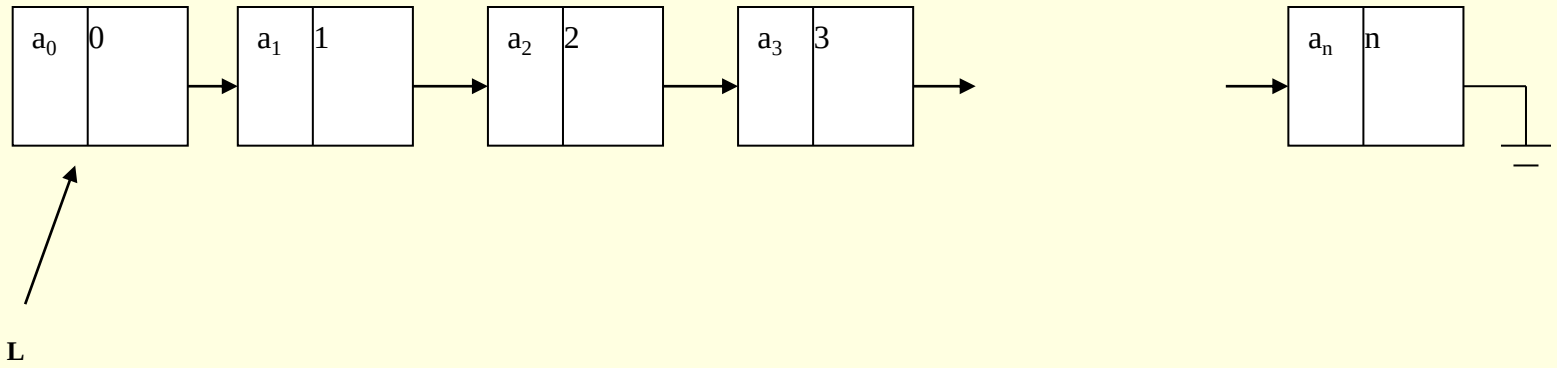
Pode ser representado como um array de tamanho n + 2, onde n é o grau:

$(n, a_n, a_{n-1}, ..., a_1, a_0)$

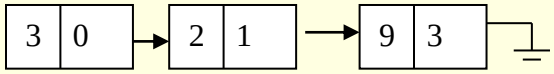
Mas o que fazer com polinômios esparsos?

$X^{1000} - 1$????

Podem ser armazenados numa lista encadeada da forma:



Ex: $p(x) = 9x^3 + 2x + 3$



No caso do polinômio, assuma que os termos são dados geralmente em ordem crescente de expoente, para facilitar as operações, (como no caso dos inteiros longos).

Algumas operações relacionadas

- void soma (apont p, apont q);
  Recebe dois polinômios, p e q, e devolve em p a soma de p e q.
- void subtrai (apont p, apont q);
  Recebe dois polinômios, p e q, e devolve em p a diferença entre p e q. ou seja, p-q.
- void multiplica (apont p, apont q);
  Recebe dois polinômios, p e q, e devolve em p o produto de p e q.
- apont deriva (apont p);
  Recebe um polinômio p e devolve um apontador para um novo polinômio que é a derivada de p.
- int grau (apont p);
  Recebe um polinômio p e devolve o grau do polinômio.
- double calcula (apont p, double x);
  Recebe um polinômio p e um real x e devolve o valor do polinômio p em x.
- apont copia (apont p);
  Recebe um polinômio p e devolve um apontador para um novo polinômio que é uma cópia de p.
- void imprime (apont p);
  Recebe um polinômio p e o imprime.

Trabalho 2 trata de polinômios