Listas ligadas

Listas

- Tal como os vectores, as listas são contentores de objectos, todos do mesmo tipo, organizados sequencialmente (1º elemento, 2º elemento, etc)
  - Nos vectores os elementos podem ser acedidos através de índices
  - Nas listas o acesso é feito através de apontadores
  - Os vectores, uma vez criados, têm uma capacidade fixa
  - As listas são estruturas dinâmicas que reservam memória para um novo elemento apenas quando este é criado

Listas ligadas

- Todos os elementos de uma lista são do mesmo tipo, mas é possível ter listas com elementos de qualquer tipo (incluindo outras listas)
- Numa lista ligada cada elemento contém o endereço do elemento seguinte (a sequência é estabelecida explicitamente)
- Cada elemento de uma lista contém:
  - Campo(s) da informação
  - Campo do endereço do elemento seguinte

Listas ligadas

- A lista é acedida através de um ponteiro externo que aponta para o primeiro nó.
- O fim da lista é indicado pelo ponteiro para elemento seguinte a NULL

Listas ligadas

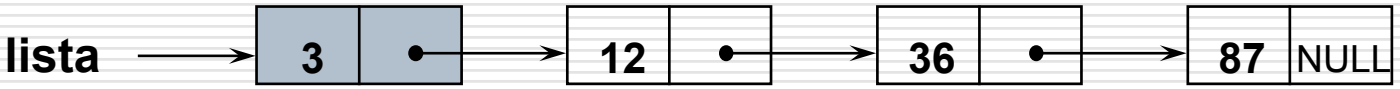
- É possível definir uma lista ligada simples como uma estrutura com as seguintes características básicas:
  - A sua dimensão é variável durante a execução de um programa, podendo crescer ou diminuir consoante as necessidades
  - Cada elemento é acedido a partir do anterior e conhece o endereço do elemento seguinte
  - A lista é acedida por um ponteiro para o seu primeiro elemento, sendo esta a forma de garantir o acesso a todos os elementos

Listas ligadas

- Em muitos casos os elementos de uma lista são mantidos ordenados em função do seu campo de informação (ou de parte dele)
  - Assim as inserções e eliminações de elementos não são feitas num ponto fixo da estrutura, mas dependem da informação a inserir e da que já se encontra na lista

Listas ligadas

- É comum que estas listas incluam um nó especial (chamado header ou cabeçalho) cuja informação não pertence realmente à lista, mas é colocado no seu início, por forma a simplificar as funções de inserção e eliminação
  - Por vezes o campo de informação do header é utilizado para manter informação relativa à própria lista, como por exemplo o seu número de elementos



Listas ligadas

- É possível definir um tipo lista ordenada com um conjunto de operações básicas
  - List cria_lista ()
    - coloca a lista acessível
  - List destroi_lista (List lista)
    - coloca a lista nacessível
  - int lista_vazia (List lista)
    - 1 se lista vazia
  - int lista_cheia (List lista)
    - sempre 0
  - void insere_lista (List lista, ITEM_TYPE item)
    - insere um elemento
  - void elimina_lista (List lista, ITEM_TYPE item)
    - elimina um elemento
  - List pesquisa_lista (List lista, ITEM_TYPE item)
    - Devolve ponteiro para o nó identificado por item (se existir)
  - void imprime_lista (List lista)
    - Imprime os campos de informação dos nós da lista

Listas ligadas

- A definição dos tipos List e List_node pode ser feita à custa de dois typedef:

```
typedef struct lnode *List;
typedef struct lnode {
    ITEM_TYPE info;
    List next;
} List_node;
```

- Esta definição tem características recursivas, pois uma lista é um apontador para uma estrutura e um dos membros dessa estrutura é uma lista
- Em cada caso será necessário redefinir ITEM_TYPE para o tipo de dados a guardar em cada nó
- É possível implementar as operações básicas do tipo List (considerando ITEM_TYPE = int)

Listas ligadas – operações básicas

- A função cria_lista resume-se à criação do header da lista e da colocação do ponteiro de lista a apontar para ele:

```
List cria_lista (void)
{
    List aux;
    aux = (List) malloc (sizeof (List_node));
    if (aux != NULL) {
        aux->info = 0;
        aux->next = NULL;
    }
    return aux;
}
```

Listas ligadas – operações básicas

- A função destroi_lista deve libertar o espaço usado pelos nós que eventualmente existam na lista

```
List destroi_lista (List lista)
{
    List temp_ptr;
    while (lista_vazia (lista) == 0) {
        temp_ptr = lista;
        lista= lista->next;
        free (temp_ptr);
    }
    free(lista);
    return NULL;
}
```

Listas ligadas – operações básicas

- A função lista_vazia deve devolver 1 se a lista estiver vazia (tem apenas o header) e 0 caso contrário

```
int lista_vazia(List lista)
{
    return (lista->next == NULL ? 1 : 0);
}
```

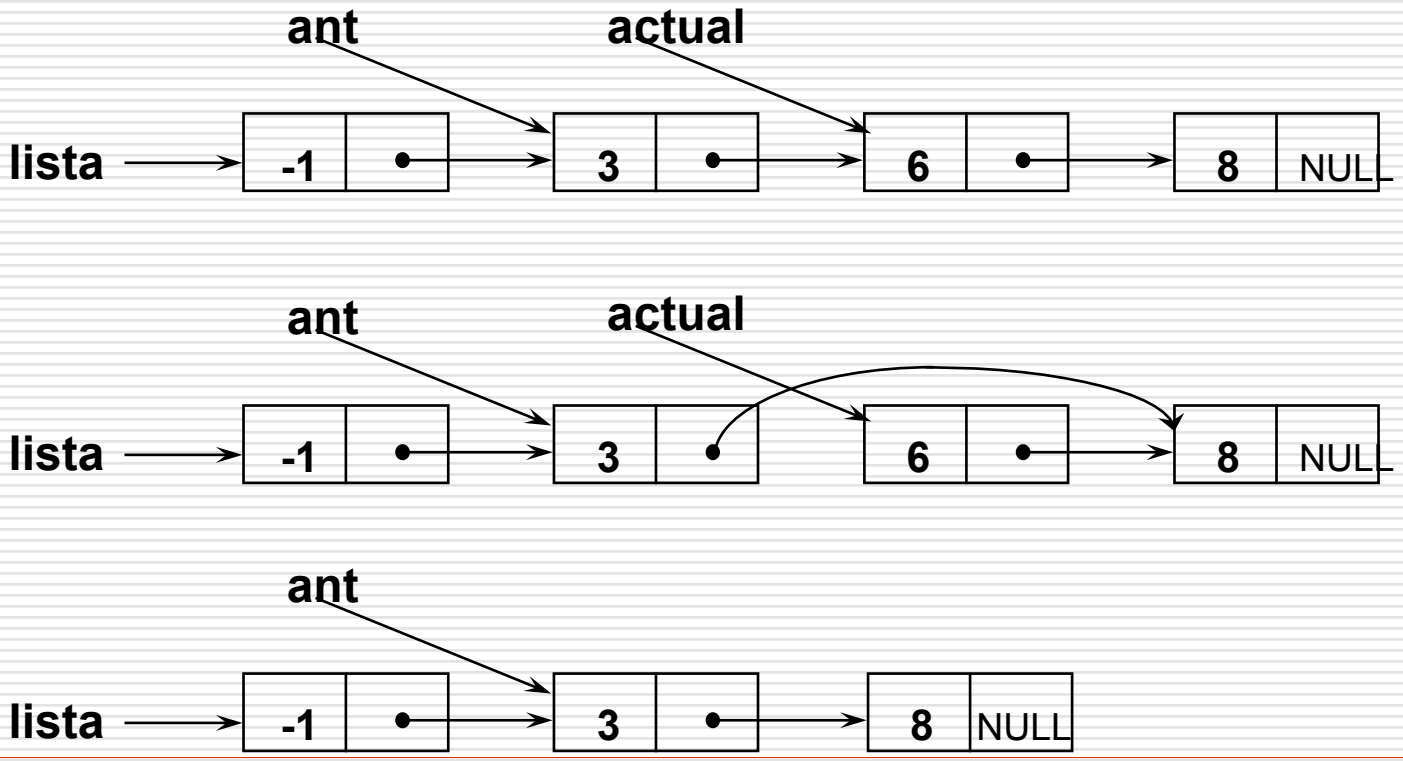
Listas ligadas – operações básicas

- Nesta implementação não há limite máximo definido para o tamanho da lista, pela que a função lista_cheia só é mantida por questões de compatibilidade, devolvendo sempre 0

```
int lista_cheia (List lista)
{
    return 0;
}
```

Listas ligadas – operações básicas

- Para eliminar um nó é preciso encontrar o nó a eliminar e o nó que o antecede. Por exemplo, para eliminar 6:



Listas ligadas – operações básicas

- Vamos começar pela função de procura do nó que se pretende eliminar e do anterior a ele:

```
void procura_lista (List lista, ITEM_TYPE chave, List *ant, List
    *actual)
{
    *ant = lista; *actual = lista->next;
    while ((*actual) != NULL && (*actual)->info < chave)
    {
            *ant = *actual;
            *actual = (*actual)->next;
    }
    if ((*actual) != NULL && (*actual)->info != chave)
            *actual = NULL;   /* Se elemento não encontrado*/
}
```

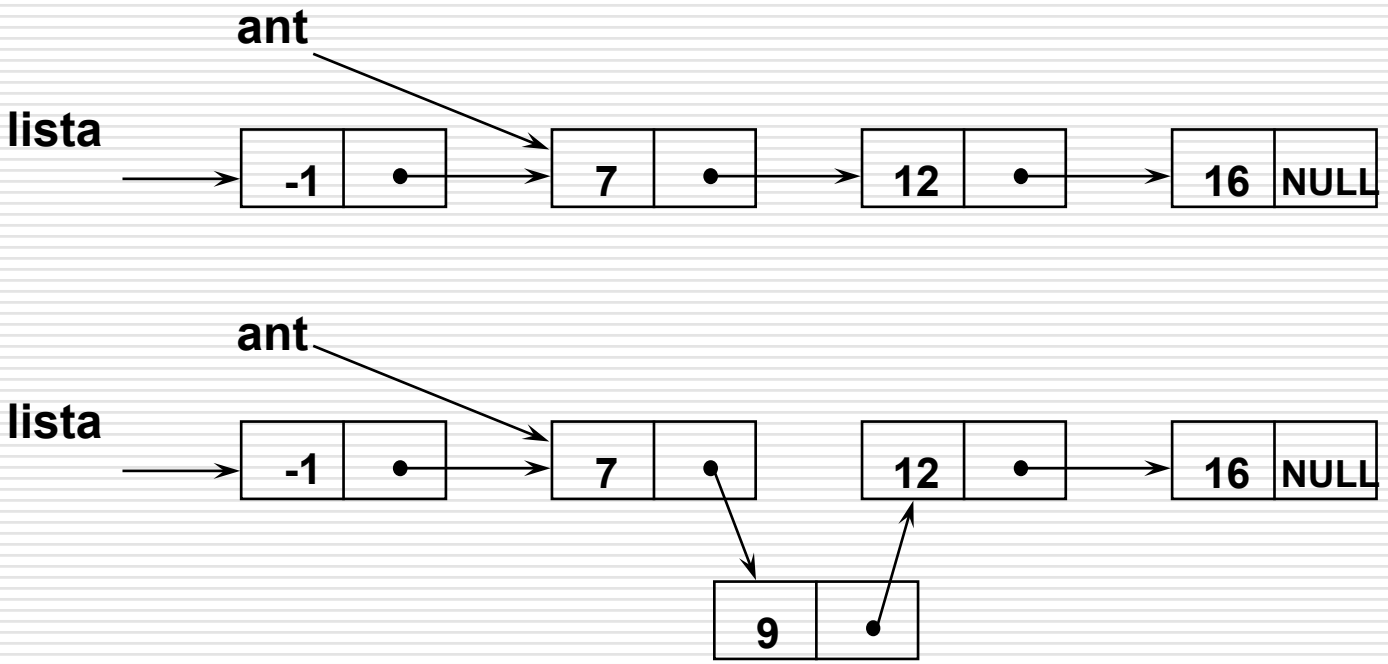
Listas ligadas – operações básicas

- A função de eliminação fica:

```
void elimina_lista (List lista, ITEM_TYPE it)
{
    List ant1;
    List actual1;
    procura_lista (lista, it, &ant1, &actual1);
    if (actual != NULL) {
            ant->next = actual->next;
            free (actual);
    }
}
```

Listas ligadas – operações básicas

- Para inserir um novo elemento e manter a ordenação é necessário encontrar o elemento a seguir ao qual se vai inserir. Por exemplo, para inserir 9:



Listas ligadas – operações básicas

- Assume-se que o elemento a inserir não existe na lista

```
void insere_lista (List lista, ITEM_TYPE it)
{
    List no;
    List ant, inutil;
    no = (List) malloc (sizeof (List_node));
    if (no != NULL) {
            no->info = it;
            procura_lista (lista, it, &ant, &inutil);
            no->next = ant->next;
            ant->next = no;
    }
}
```

Listas ligadas – operações básicas

- A função de pesquisa é implementada facilmente à custa da função auxiliar procura_lista:

```
List pesquisa_lista (List lista, ITEM_TYPE it)
{
    List ant;
    List actual;

    procura_lista (lista, it, &ant, &actual);

    return (actual);
}
```

Listas ligadas – operações básicas

- A função de impressão do conteúdo da lista também não apresenta grandes dificuldades

```
void imprime_lista (List lista)
{
    List l = lista->next;          /* Salta o header */
    while (l)
    {
        printf("%d ", l->info);
        l=l->next;
    }
}
```

Listas ligadas

- Pretende-se agora uma função que imprima os elementos da lista por ordem inversa
  - Uma versão iterativa não é simples e eficáz
  - É fácil encontrar o último elemento da lista, mas para encontrar o penúltimo será necessário voltar ao início da lista e percorrê-la toda
  - O mesmo para os restantes elementos

Listas ligadas

- Nesta situação uma solução recursiva é mais adequada (as soluções recursivas são muitas vezes adequadas para manipular estruturas de dados dinâmicas)

```
void imprime_contrario (List lista)
{
    if (lista->next == NULL)
       printf ("%d ", lista->info);
    else {
                imprime_contrario (lista->next);
                printf("%d ", lista->info);
            }
 }
```

Listas ligadas

- No entanto, esta solução imprime também o campo de informação do header, o que não é correcto
  - Esta situação pode ser evitada com uma nova função que se limita a chamar a anterior
  - Esta nova função é que deve ser chamada pelos programas que pretendem escrever o conteúdo de uma lista por ordem inversa

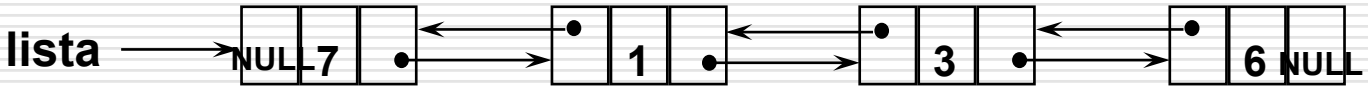
```
void imprime_lista_inverte (List lista)
{
    imprime_contrario(lista->next);
}
```

Listas duplamente ligadas

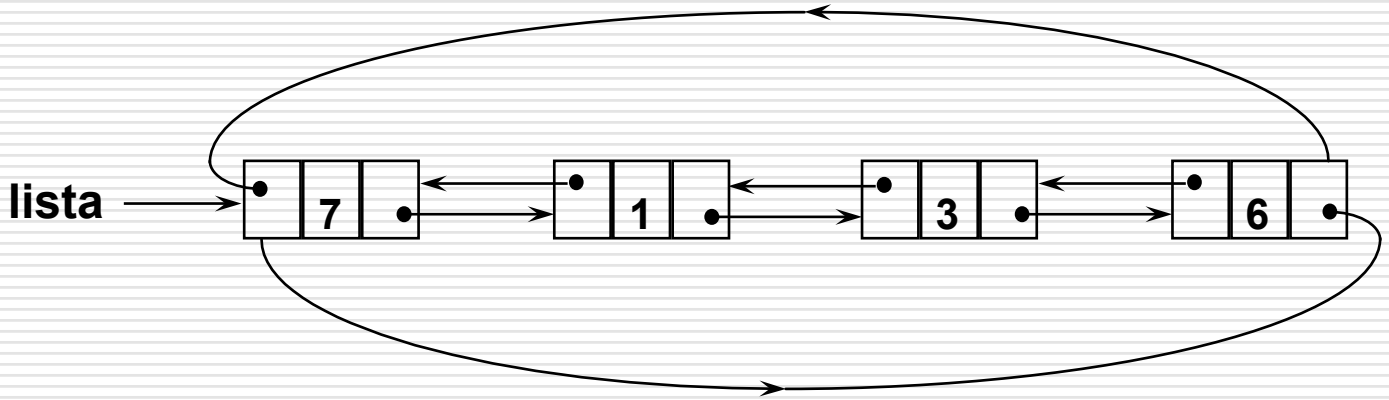
- Um dos inconvenientes das listas ligadas é o facto da sua travessia só poder ser feita num sentido (do início para o fim)
- Esse problema é ultrapassado pelas listas duplamente ligadas, as quais podem ainda ser circulares (tal como as listas simples)
- As listas duplamente ligadas podem ser atravessadas nos dois sentidos, pois cada nó tem um apontador para o elemento seguinte e um apontador para o elemento anterior

Listas duplamente ligadas

Simples



Circular



Listas duplamente ligadas

- Cada nó destas listas tem, pelo menos, três campos:
  - campo (ou campos) para a informação
  - ponteiro para o nó anterior
  - ponteiro para o nó seguinte
- Uma declaração possível (no caso da informação ser int) é:

```
typedef struct lnode2 *List2;
typedef struct lnode2 {
    int info;
    List2 next;
    List2 previous
} List_node2;
```

Listas duplamente ligadas

- As operações básicas sobre o tipo lista duplamente ligada são semelhantes às das listas ligadas simples
  - Na implementação haverá que ter em conta que cada nó contém apontadores para dois nós e fazer os ajustes necessários

Listas duplamente ligadas

- Por exemplo, a operação de eliminação de um nó fica

```
void elimina_listaDupla (List2 listaD, int it)
{
    List2 ant, actual, temp;
    procura_lista2 (listaD, it, &ant, &actual);
    if (actual != NULL) {
        ant ->next = actual->next;
        temp= actual->next;
        temp->previous = ant;
        free (actual);
    }
}
```