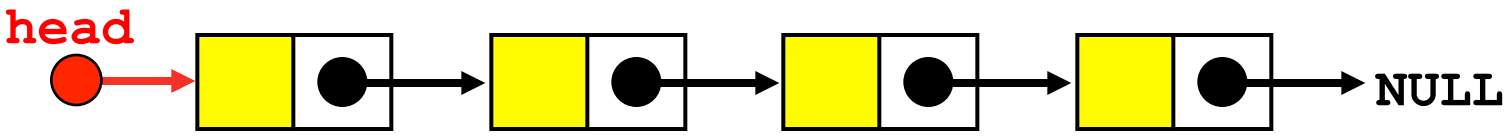
DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Dicas para o projecto *2*

IAED, 2013/2014

Ideia geral



Uma fila FIFO pode ser implementada através de uma lista ligada:

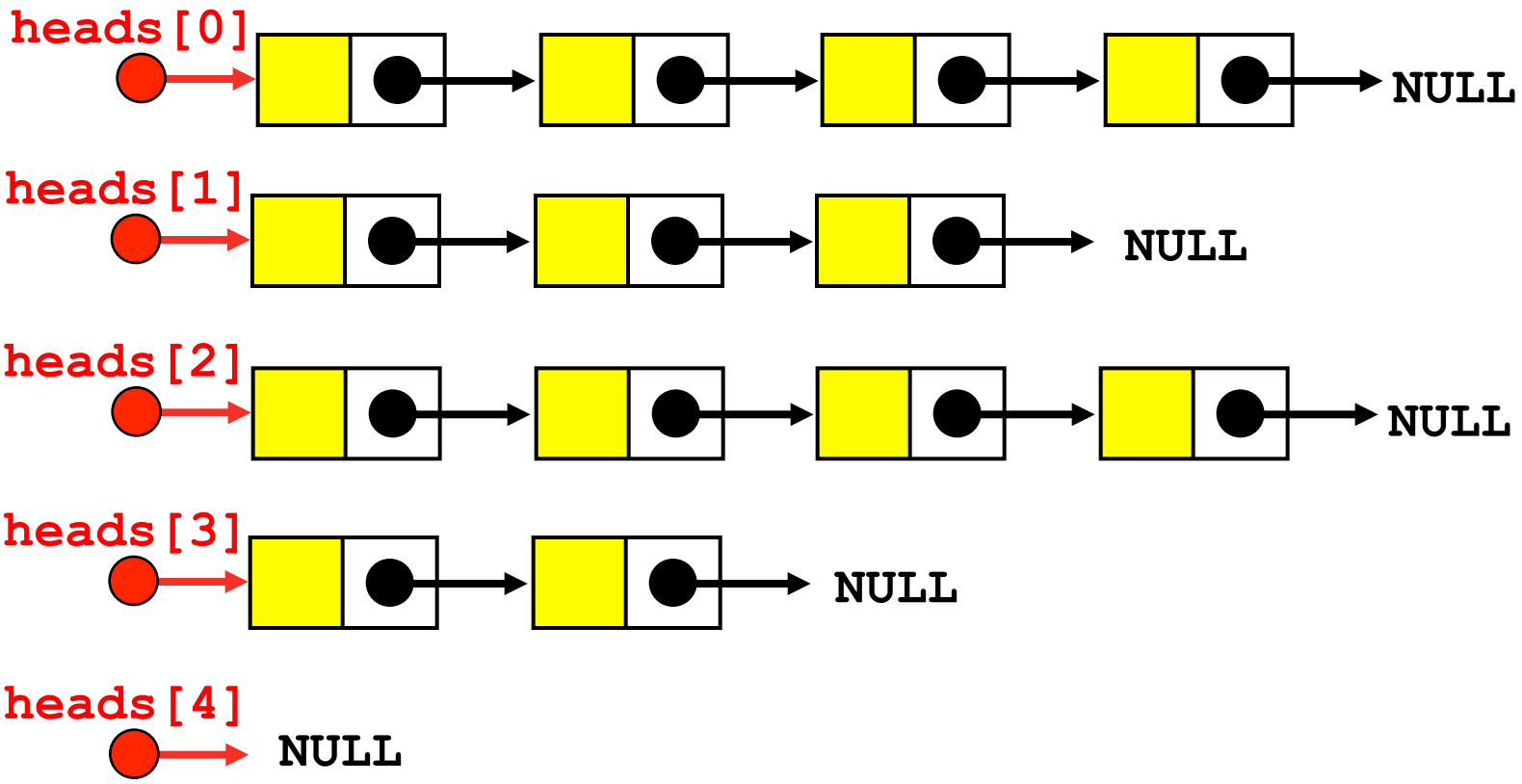
Inserimos no fim, tiramos do início!

Neste caso, temos N utilizadores/agentes, cada um com uma fila de espera (FIFO).

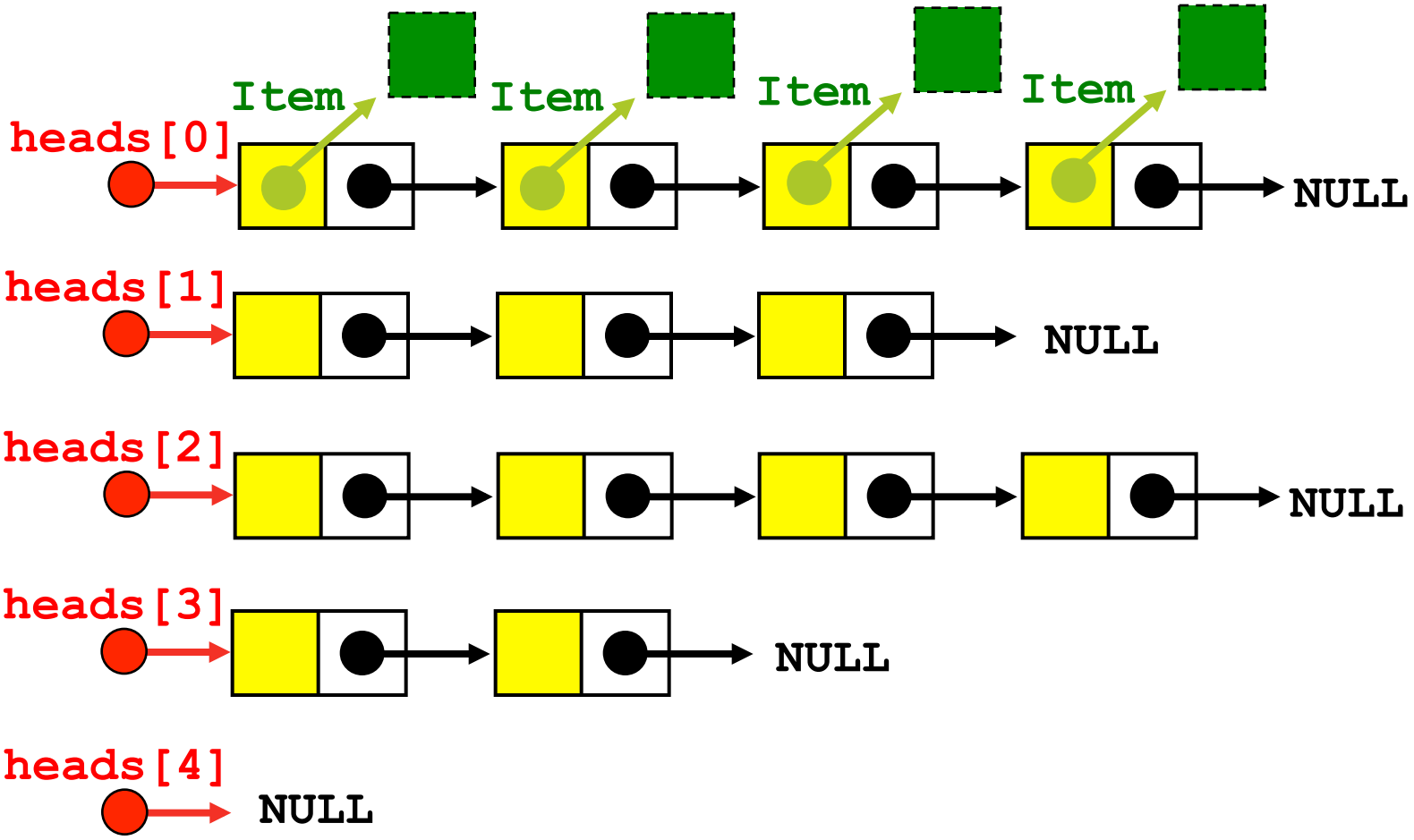
Uma forma de implementar será ter um "vector de heads". Sempre que queremos acrescentar uma mensagem ao receptor i vamos à lista de índice i

Ideia geral

Criamos um vector de listas

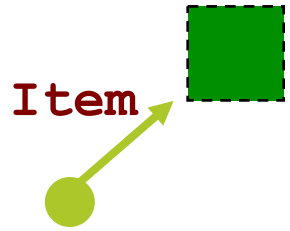


Ideia geral: onde guardamos as mensagens?

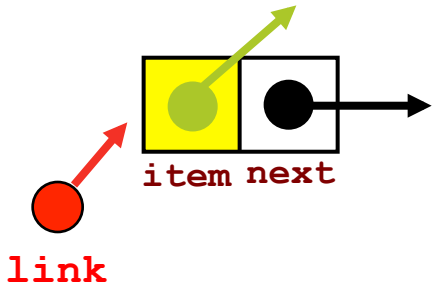


Estruturas úteis

```
typedef struct mensagem{
(emissor, receptor, mensagem…)
}*Item;
```

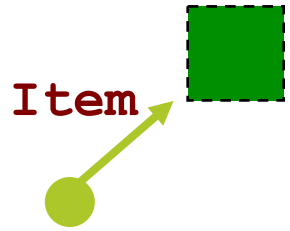



```
typedef struct node{
   Item item;
   struct node*next;
}*link;
```

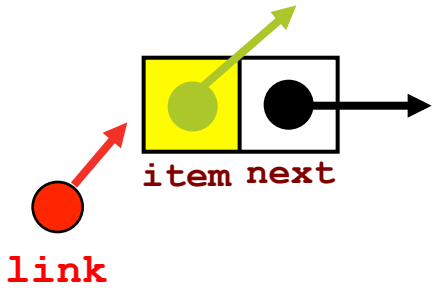


Estruturas úteis

```
typedef struct mensagem{
(emissor, receptor, mensagem…)
}*Item;
```

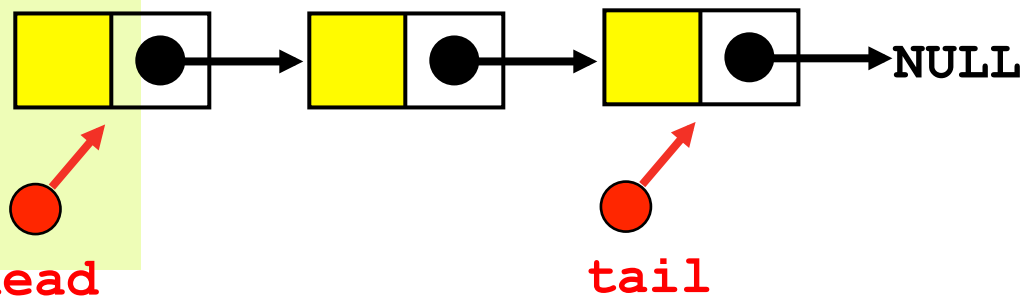



```
typedef struct node{
   Item item;
   struct node*next;
}*link;
```



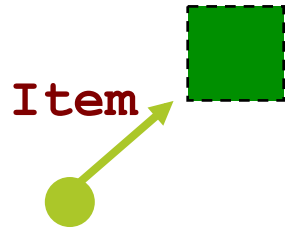
Esta também pode dar jeito… Neste caso teriam um vector de Queue's e não de heads

```
typedef struct queue{
   link head, tail;
}*Queue;
```

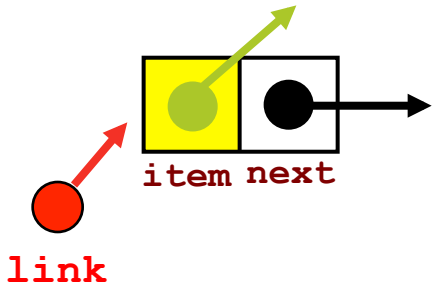


Estruturas úteis

```
typedef struct mensagem{
(emissor, receptor, mensagem…)
}*Item;
```

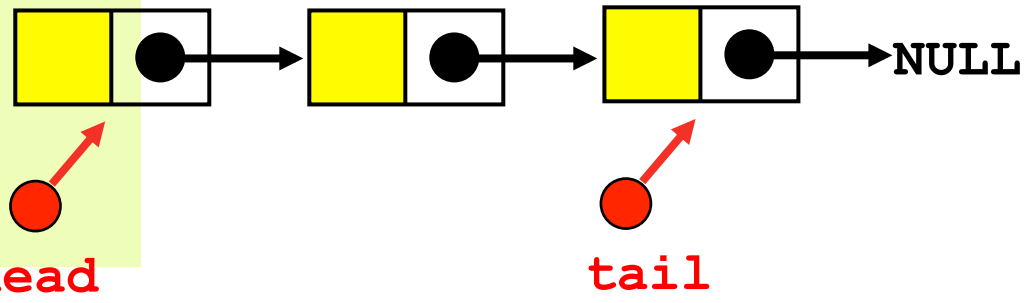



```
typedef struct node{
   Item item;
   struct node*next;
}*link;
```



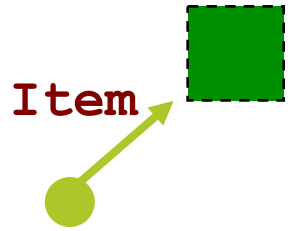
Esta também pode dar jeito… Neste caso teriam um vector de Queue's e não de heads

```
typedef struct queue{
   link head, tail;
   int size;
}*Queue;
```

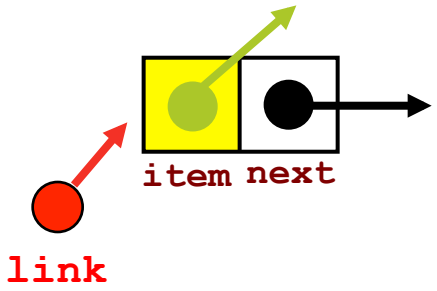


Mais abstracção

```
typedef struct mensagem{
(…)
}*Item;
```

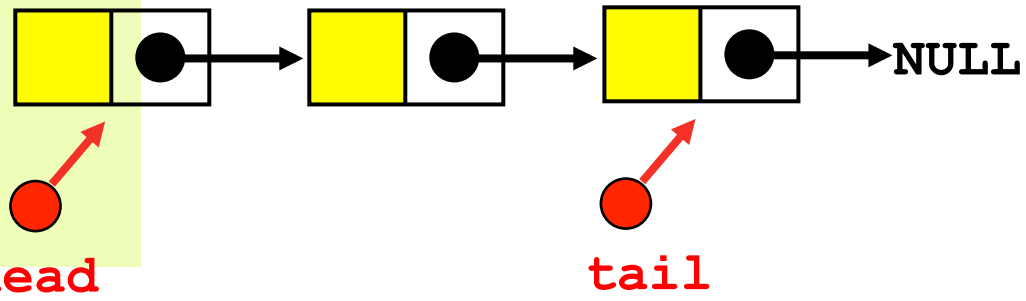



```
typedef struct node{
   void* item;
   struct node*next;
}*link;
```



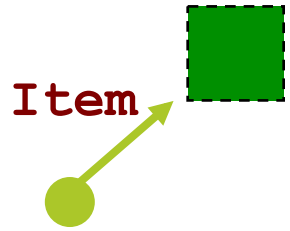
Esta também pode dar jeito… Neste caso teriam um vector de Queue's e não de heads

```
typedef struct queue{
   link head, tail;
   int size;
}*Queue;
```



Funções úteis – Item's

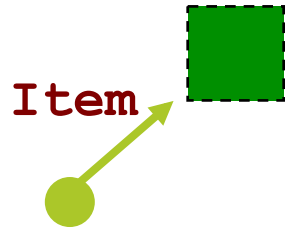
```
typedef struct mensagem{
   (…)
}*Item;
```




```
Item NewItem (char*message, int sender, int receiver)
{
   /* cria um novo Item */
}
```

Funções úteis – Item's

```
typedef struct mensagem{
   (…)
}*Item;
```

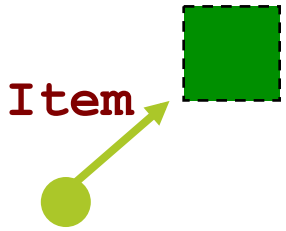



```
void deleteItem (Item x)
{
   /* liberta a memoria alocada */
}
```

```
void showItem(Item x)
{
  /* mostra o conteúdo de um item – ver enunciado */
}
```

Funções úteis – Item's

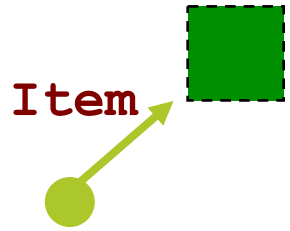
```
typedef struct mensagem{
   (…)
}*Item;
```




```
int cmpItem(Item a, Item b)
{
   /* retorna um valor < 0 se a<b, 0 se forem iguais e um valor
   >0 se b>a */
}
```

Funções úteis – Item's

```
typedef struct mensagem{
   (…)
}*Item;
```

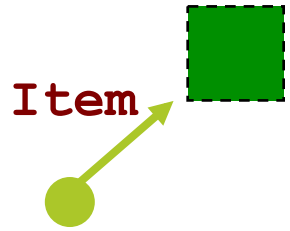



```
int cmpItem(const void *a, const void* b)
{
   /* retorna um valor < 0 se a<b, 0 se forem iguais e um valor
   >0 se b>a */
}
```

Se definir de uma forma geral (com void*) pode tentar usar a função **qsort** da **stdlib.h**....

Funções úteis – Item's

```
typedef struct mensagem{
   (…)
}*Item;
```

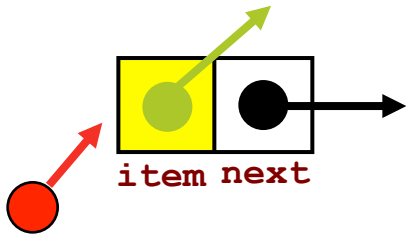



```
Item newItem(char*s, int sender, int receiver);
void deleteItem(Item a);
void showItem(Item a);
int cmpItem(Item a, Item b);
void sort(Item a[], int l, int r);
```

Sugestão: ver https://web.ist.utl.pt/~ist14152/aed/wiki.cgi/download/AllSortsDotC

FIFO's | funções úteis | node's & links

```
typedef struct node{
   Item item; /* ou void*item*/
   struct node*next;
}*link;
```



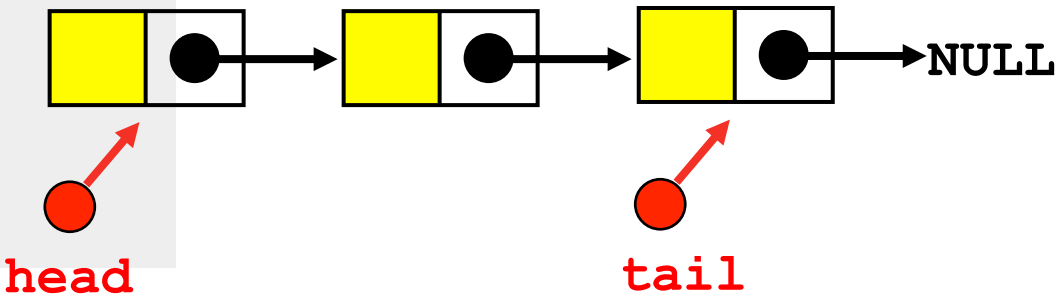
Funções úteis.

Para uma dada queue/lista, procure criar uma função que

- Inicialize uma dada lista/queue.
- Remova todos os elementos de uma lista libertando a memoria associada.
- Insira um novo elemento.
- Devolva o item do primeiro elemento e elimine o nodo respectivo.
- Liste todos os Items da lista
- Crie um vector com todos os elementos da lista (*).

Para o 20! ☺

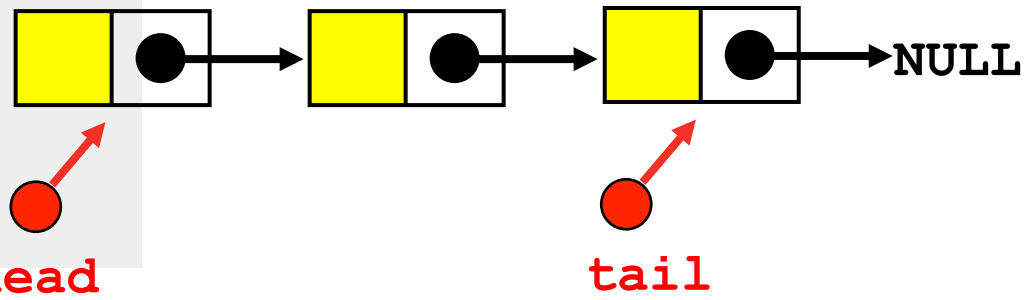
```
typedef struct queue{
   (…);
}*Q;
```



- Os comandos process e send não devem escalar com o tamanho da lista/queue …
- Analise a escalabilidade do seu código quando
  1. Insere/processa uma mensagem
  2. ordena os elementos da lista

Para o 20! ☺

```
typedef struct queue{
   (…);
}*Q;
```



Procure usar abstrações (ADTs)!

Organize o código da melhor forma, se possível em vários ficheiros.

Faça uma utilização racional da memória.

Documente convenientemente o seu código!

Elimine eventuais fugas de memória (`valgrind`).

Bons códigos!