

Listas Ligadas I

25/10/2023

Ficheiros com exemplos

- Está disponível no Moodle um **ficheiro ZIP** de suporte aos tópicos de hoje
- Implementação de tipos abstratos usando uma **lista ligada** como representação interna
- Exemplos simples de aplicação
- **Implementações incompletas**, que permitem trabalho autónomo de desenvolvimento e teste

Sumário

- Recap
- O TAD **STACK** – usando uma lista ligada
- O TAD **QUEUE** – usando uma lista ligada
- O TAD **LIST** – funcionalidades principais

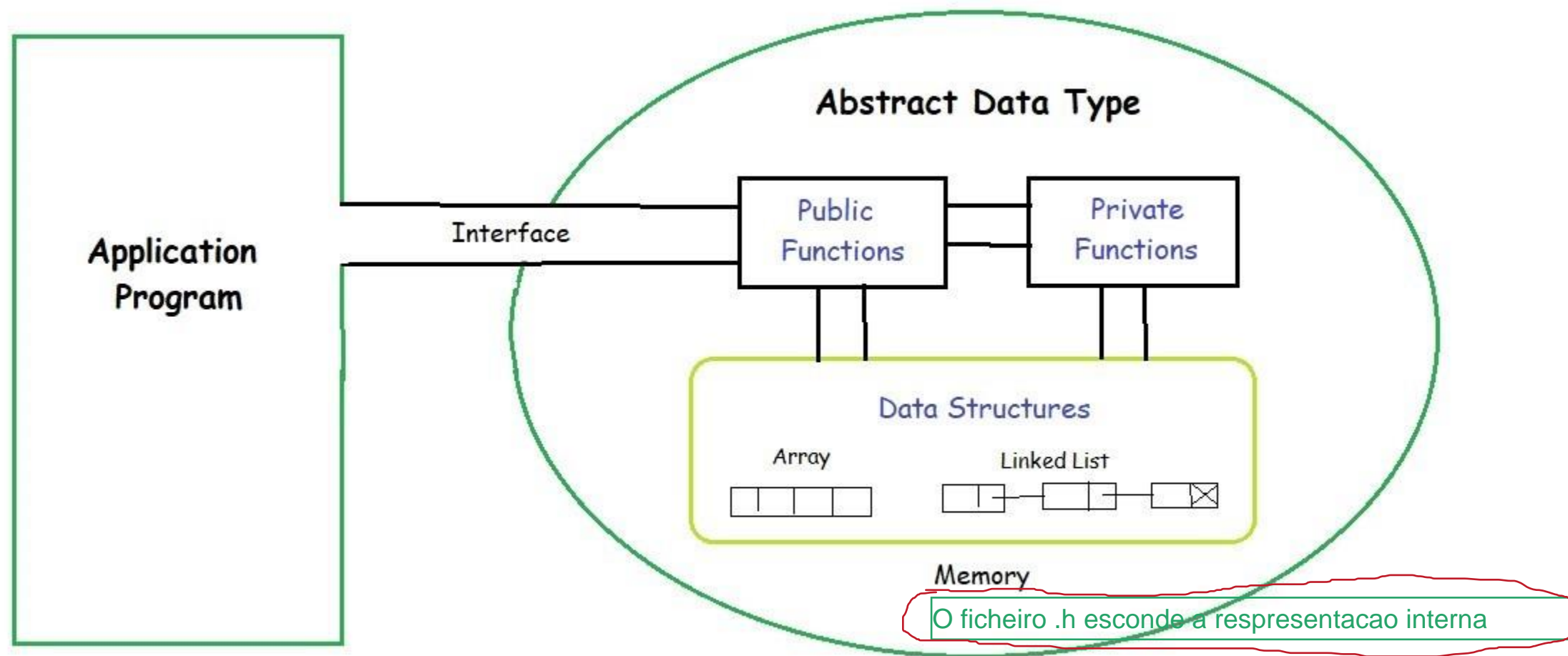
Vamos focar nas funcionalidades numa lista,

ADT - Abstract Data Type
TAD - Tipo Abstrato de Dados

Let's
RECAP

Recapitulação

Tipo Abstrato de Dados (TAD)



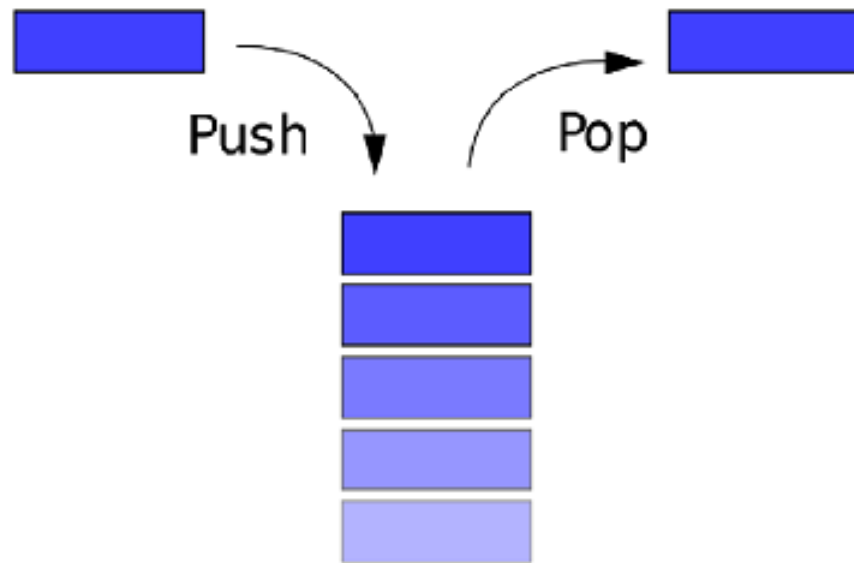
[geeksforgeeks.org]

Tipo Abstrato de Dados (TAD)

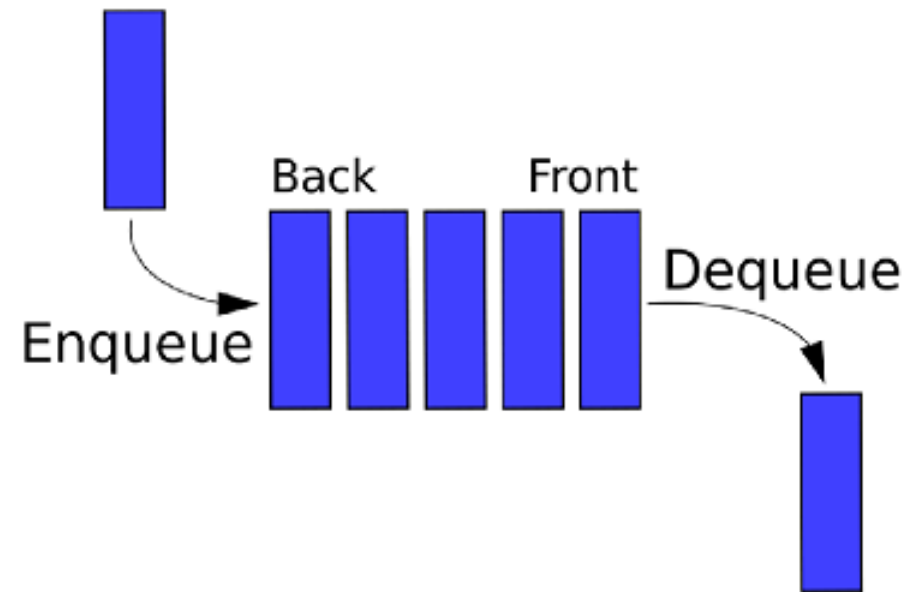
- TAD = especificação + interface + implementação
- Encapsular detalhes da representação / implementação
- Flexibilizar manutenção / reutilização / portabilidade

- Ficheiro .h : operações públicas + ponteiro para instância
- Ficheiro .c : implementação + representação interna

STACK e QUEUE



Stack (LIFO) last in first out



Queue (FIFO) first in first out

[github.io]

- Num próximo **guião prático** iremos usar / aplicar

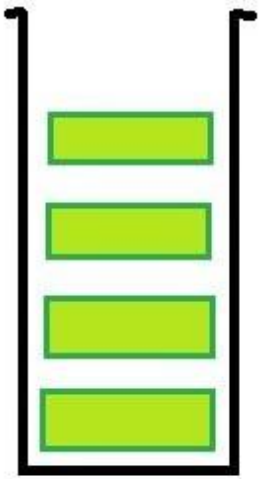
PointersStack.h

```
#ifndef _POINTERS_STACK_  
#define _POINTERS_STACK_  
  
typedef struct _PointersStack Stack;  
  
Stack* StackCreate(int size);  
  
void StackDestroy(Stack** p);  
  
void StackClear(Stack* s);  
  
int StackSize(const Stack* s);  
  
int StackIsFull(const Stack* s);  
  
int StackIsEmpty(const Stack* s);  
  
void* StackPeek(const Stack* s);  
  
void StackPush(Stack* s, void* p);  
  
void* StackPop(Stack* s);  
  
#endif // _POINTERS_STACK_
```

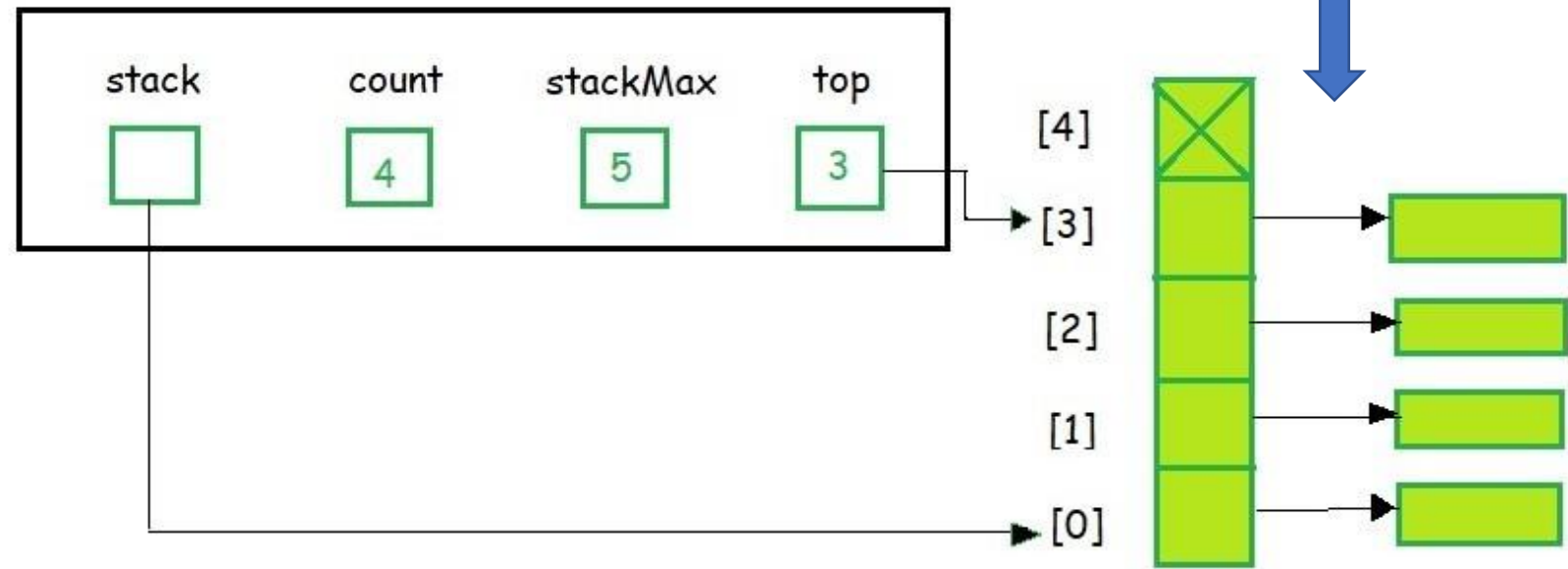


O TAD Stack – Array de ponteiros

a) Conceptual



b) Physical Structure



PointersQueue.h



```
#ifndef _POINTERS_QUEUE_
#define _POINTERS_QUEUE_

typedef struct _PointersQueue Queue;

Queue* QueueCreate(int size);

void QueueDestroy(Queue** p);

void QueueClear(Queue* q);

int QueueSize(const Queue* q);

int QueueIsFull(const Queue* q);

int QueueIsEmpty(const Queue* q);

void* QueuePeek(const Queue* q);

void QueueEnqueue(Queue* q, void* p);

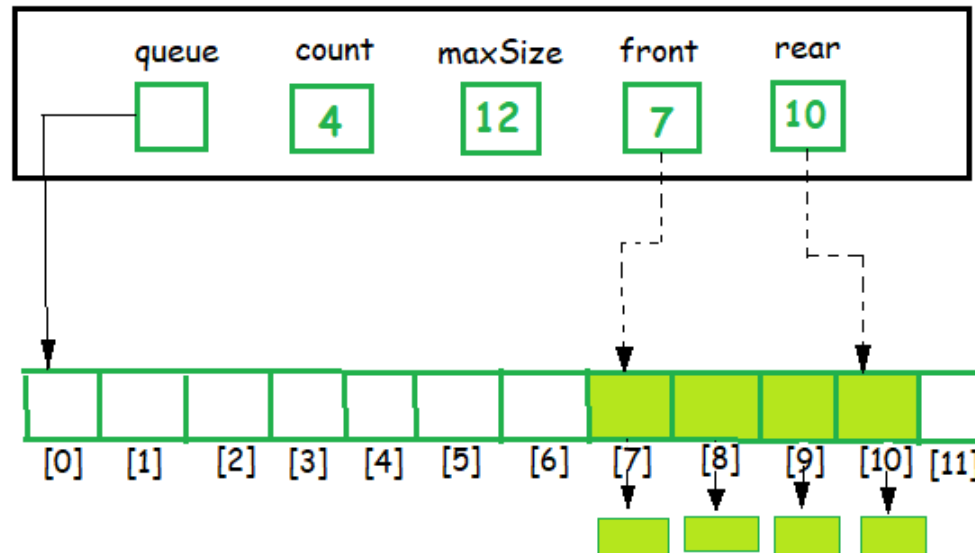
void* QueueDequeue(Queue* q);

#endif // _POINTERS_QUEUE_
```

O TAD QUEUE – **Array circular** de ponteiros



a) Conceptual



b) Physical Structures

DEQUE – Tentaram fazer ? – Questões ?



[java2novice.com]

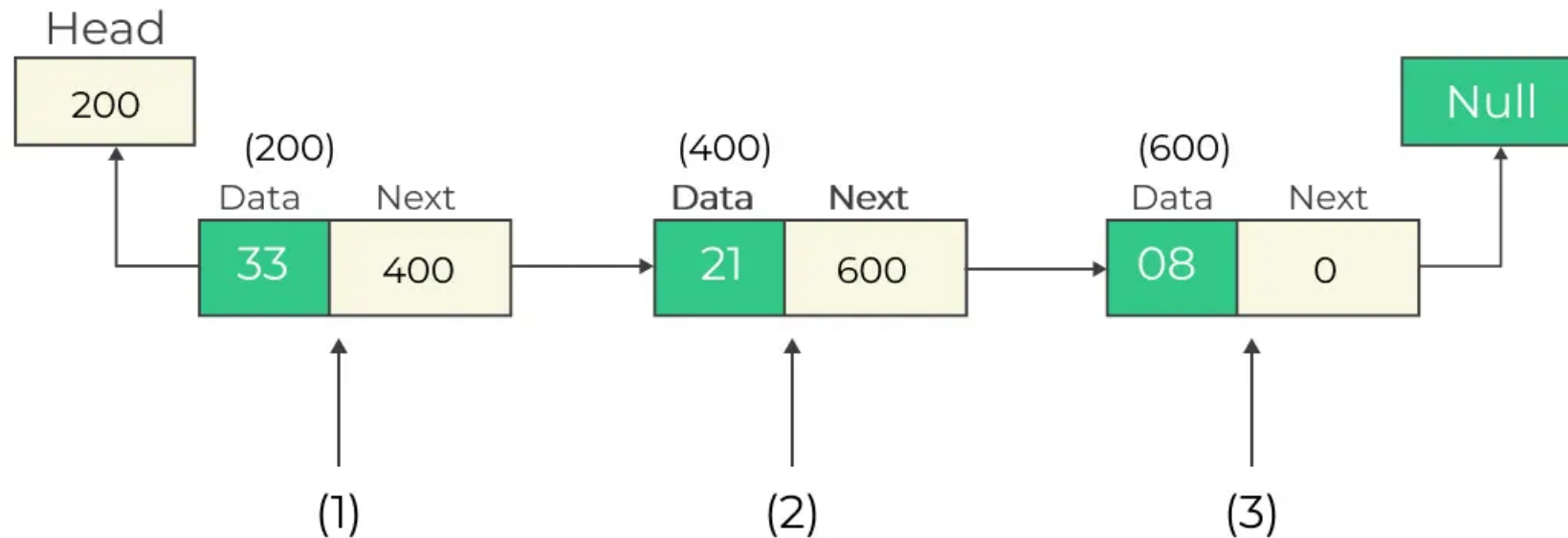
O TAD STACK / PILHA

- Lista de Ponteiros Genéricos



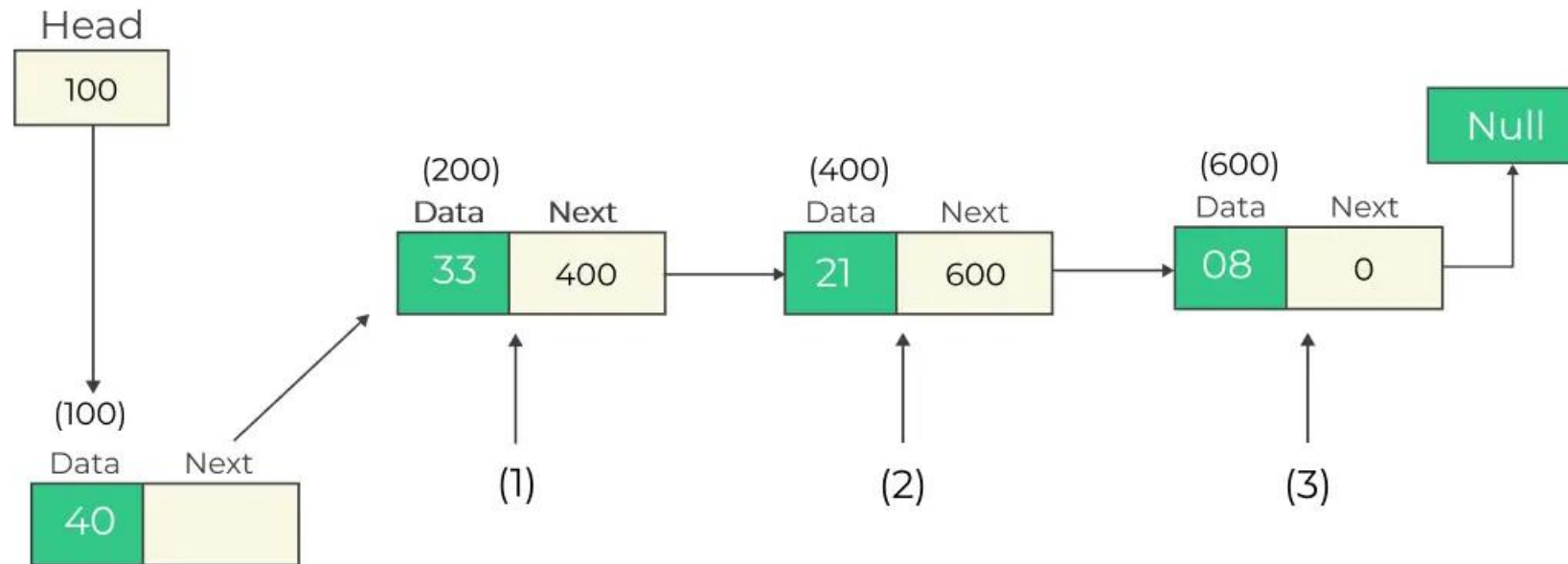
[Wikipedia]

Stack usando uma **lista ligada**



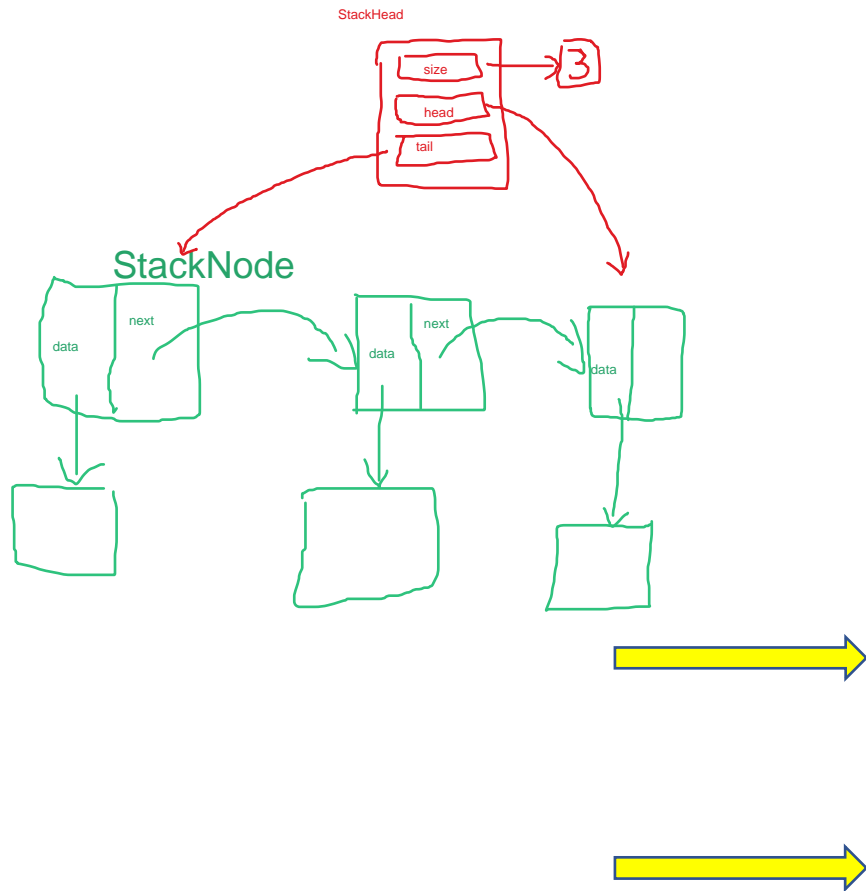
[prepinsta.com]

Stack usando uma lista ligada – push(40)



[prepinsta.com]

PointersStack.h



```
#ifndef _POINTERS_STACK_
#define _POINTERS_STACK_

typedef struct _PointersStack Stack;

Stack* StackCreate(int size);

void StackDestroy(Stack** p);

void StackClear(Stack* s);

int StackSize(const Stack* s);

int StackIsFull(const Stack* s);

int StackIsEmpty(const Stack* s);

void* StackPeek(const Stack* s);

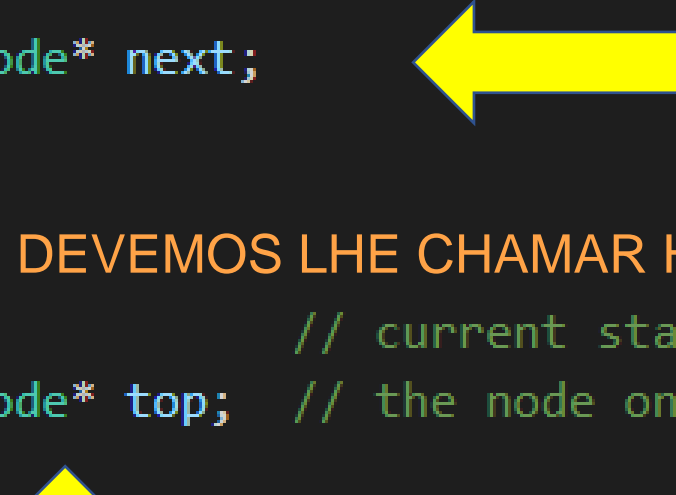
void StackPush(Stack* s, void* p);

void* StackPop(Stack* s);

#endif // _POINTERS_STACK_
```


O TAD Stack – Usando uma **lista ligada**


```
struct _PointersStackNode {  
    void* data;  
    struct _PointersStackNode* next;  
};  
  
struct _PointersStack {  
    int cur_size;           // current stack size  
    struct _PointersStackNode* top; // the node on the top of the stack  
};
```




PointersStack.c

Como nós queremos apagar o ponteiro, logo s

```
Stack* StackCreate(void) {  
    Stack* s = (Stack*)malloc(sizeof(Stack));  
    assert(s != NULL);  
  
    s->cur_size = 0;  
    s->top = NULL;  
    return s;  
}
```



```
void StackDestroy(Stack** p) {  
    assert(*p != NULL);  
    Stack* s = *p;  
  
    StackClear(s);  
  
    free(s);  
    *p = NULL;  
}
```



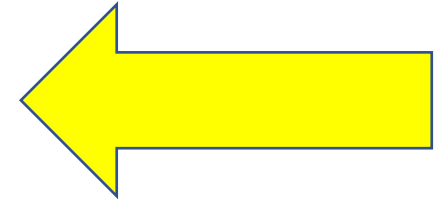
PointersStack.c

```
void StackClear(Stack* s) {  
    assert(s != NULL);  
  
    struct _PointersStackNode* p = s->top;  
    struct _PointersStackNode* aux;  
  
    while (p != NULL) {  
        aux = p;  
        p = aux->next; P == NULL -> next == NULL  
        free(aux);  
    }  
  
    s->cur_size = 0;  
    s->top = NULL;  
}
```

Temos de ter cuidado para não perder acesso aos recursos

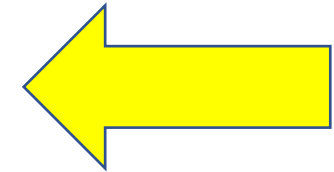
PointersStack.c

```
void StackPush(Stack* s, void* p) {  
    assert(s != NULL);  
  
    struct _PointersStackNode* aux;    criamos o nó  
    aux = (struct _PointersStackNode*)malloc(sizeof(*aux));  
    assert(aux != NULL);  
  
    aux->data = p;  
    aux->next = s->top;  
  
    null  
    s->top = aux;  
  
    s->cur_size++;  
}
```



PointersStack.c

```
void* StackPop(Stack* s) {  
    assert(s != NULL && s->cur_size > 0);  
  
    struct _PointersStackNode* aux = s->top;  
    s->top = aux->next;  
    s->cur_size--;  
  
    void* p = aux->data;  
  
    free(aux);  
  
    return p;  
}
```

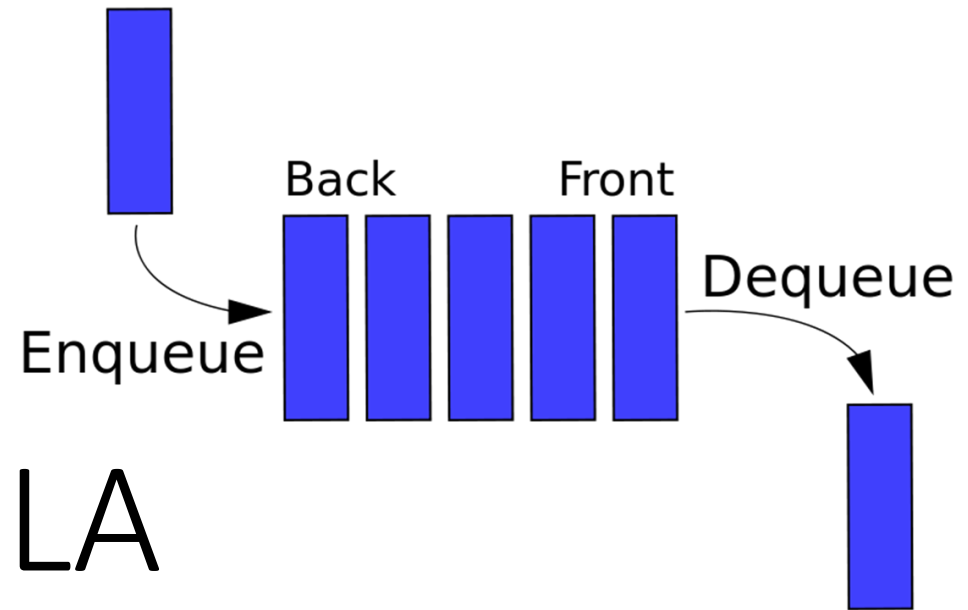


PointersStack.h + PointersStack.c

- **TAREFA** : Analisar a **implementação** das funções do TAD
- Analisar / Estudar a implementação das **operações** sobre a estrutura de dados **lista ligada** !

Aplicação – Escrever pela ordem inversa

- Já sabemos como escrever pela **ordem inversa** os **algarismos** de um **número** inteiro positivo
- São necessárias **modificações** no código do exemplo para se utilizar esta **nova versão** do **TAD STACK** ?
- **TAREFA** : Analisar o exemplo de aplicação !!



[Wikipedia]

O TAD QUEUE / FILA

- Lista de Ponteiros Genéricos

Queue usando uma **lista ligada**

Adding the elements into Queue



Removing the elements from Queue



Printing the Queue



prepinsta.com]

PointersQueue.h



```
#ifndef _POINTERS_QUEUE_
#define _POINTERS_QUEUE_

typedef struct _PointersQueue Queue;

Queue* QueueCreate(int size);

void QueueDestroy(Queue** p);

void QueueClear(Queue* q);

int QueueSize(const Queue* q);

int QueueIsFull(const Queue* q);

int QueueIsEmpty(const Queue* q);

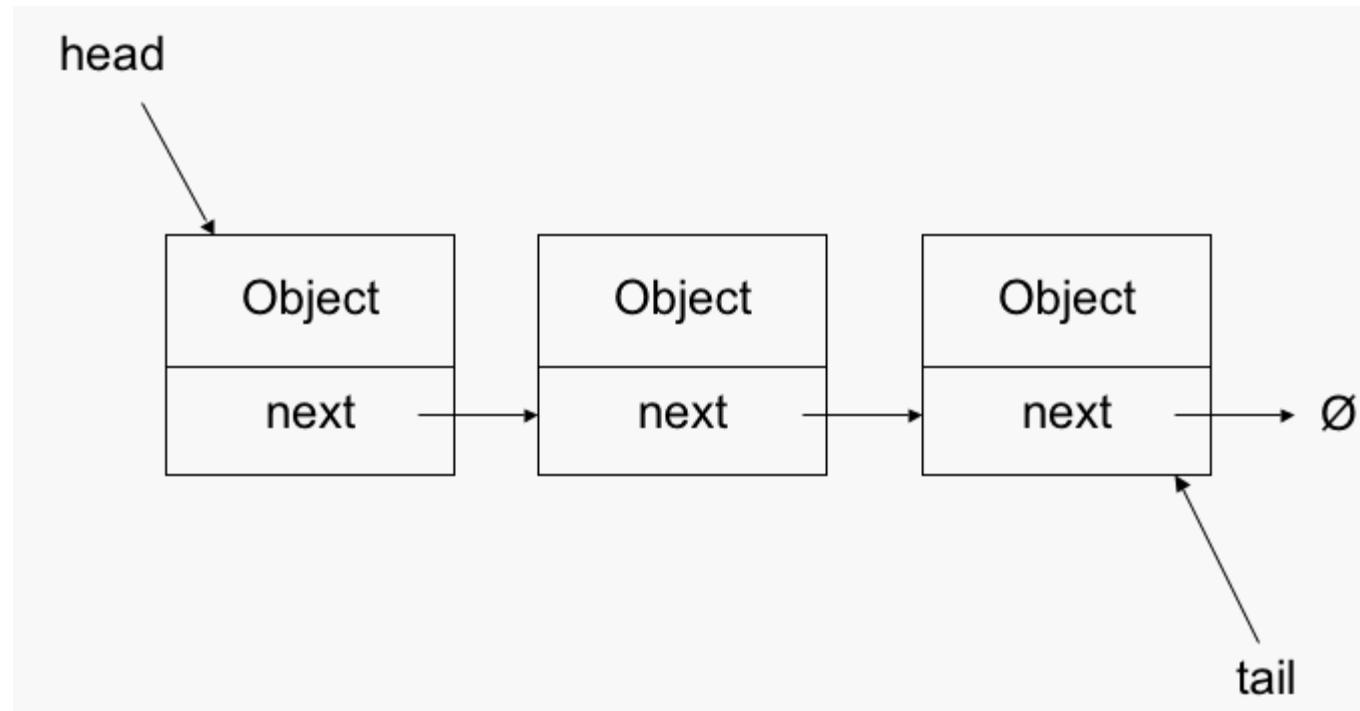
void* QueuePeek(const Queue* q);

void QueueEnqueue(Queue* q, void* p);

void* QueueDequeue(Queue* q);

#endif // _POINTERS_QUEUE_
```

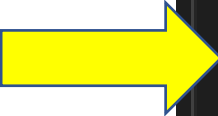
Lista Ligada – Acesso ao início e ao fim da lista



[usfCS]

O TAD QUEUE – Usando uma **lista ligada**

```
struct _PointersQueueNode {  
    void* data;  
    struct _PointersQueueNode* next;  
};  
  
struct _PointersQueue {  
    int size; // current Queue size  
    struct _PointersQueueNode* head; // the head of the Queue  
    struct _PointersQueueNode* tail; // the tail of the Queue  
};
```



PointersQueue.c

```
Queue* QueueCreate(void) {  
    Queue* q = (Queue*)malloc(sizeof(Queue));  
    assert(q != NULL);  
  
    q->size = 0;  
    q->head = NULL;  
    q->tail = NULL;  
    return q;  
}
```

```
void QueueDestroy(Queue** p) {  
    assert(*p != NULL);  
    Queue* q = *p;  
  
    QueueClear(q);  
  
    free(q);  
    *p = NULL;  
}
```

PointersQueue.c

```
void QueueEnqueue(Queue* q, void* p) {
    assert(q != NULL);

    struct _PointersQueueNode* aux;
    aux = (struct _PointersQueueNode*)malloc(sizeof(*aux));
    assert(aux != NULL);

    aux->data = p;
    aux->next = NULL;

    q->size++;

    if (q->size == 1) {
        q->head = aux;
        q->tail = aux;
    } else {
        q->tail->next = aux;
        q->tail = aux;
    }
}
```

```
graph LR
    aux[aux] -- next --> tail[ ]
    tail -- next --> next_node[ ]
    next_node -- next --> NULL[NULL]
```

PointersQueue.c

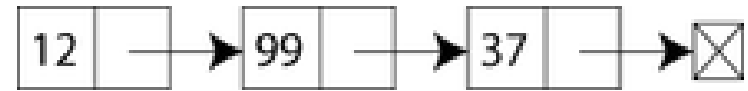
```
void* QueueDequeue(Queue* q) {  
    assert(q != NULL && q->size > 0);  
  
    struct _PointersQueueNode* aux = q->head;  
    void* p = aux->data;  
  
    q->size--;  
  
    if (q->size == 0) {  
        q->head = NULL;  
        q->tail = NULL;  
    } else {  
        q->head = aux->next;  
    }  
  
    free(aux);  
  
    return p;  
}
```

PointersQueue.h + PointersQueue.c

- **TAREFA** : Analisar a **implementação** das funções do TAD
- Analisar / Estudar a implementação das **operações** sobre a estrutura de dados **lista ligada** !

Aplicação – Testar o funcionamento do TAD

- **TAREFA** : Analisar o exemplo de aplicação !!



[Wikipedia]

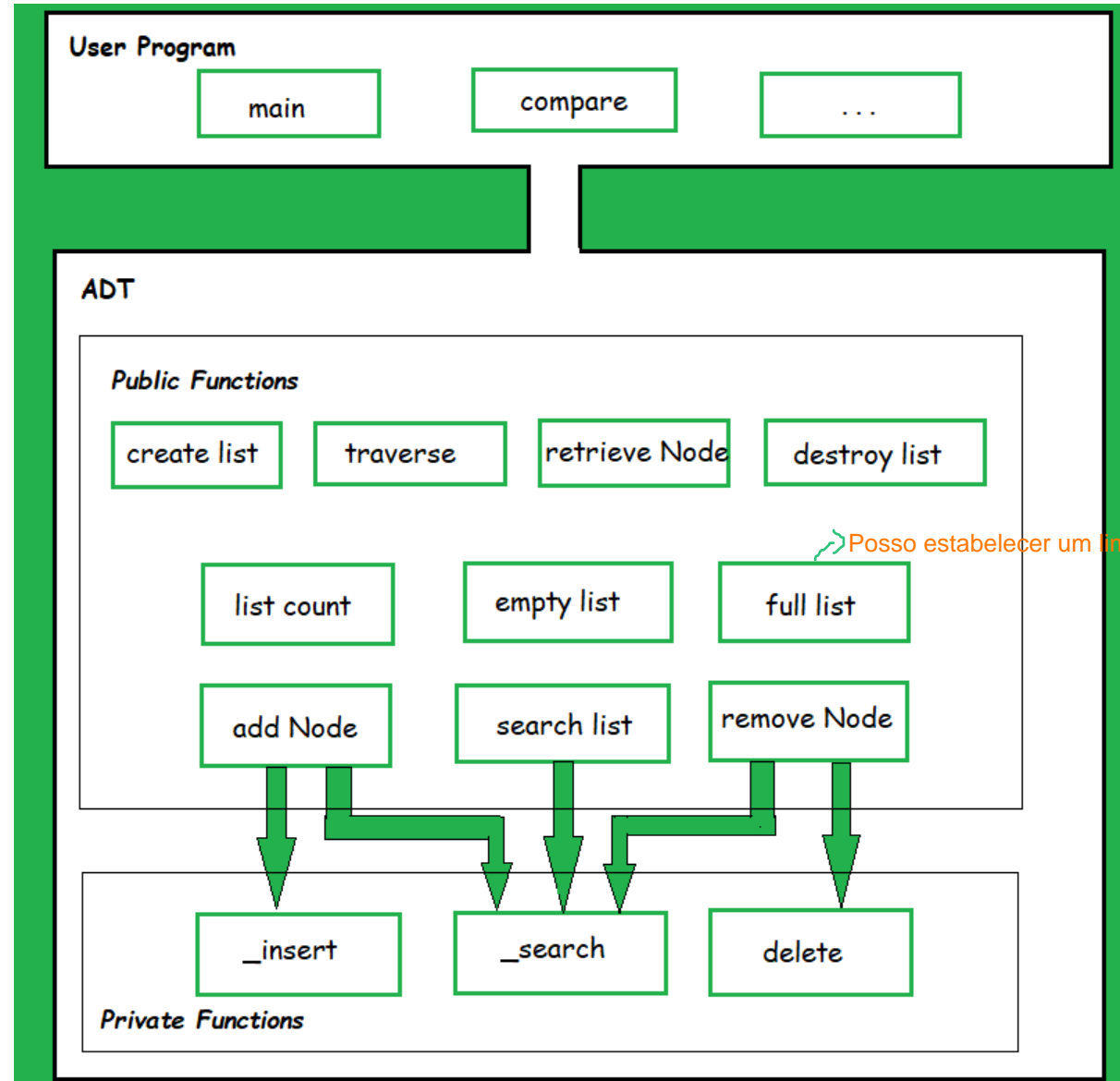
O TAD **LISTA**

Lista sem ordem! Podemos adicionar no meio!

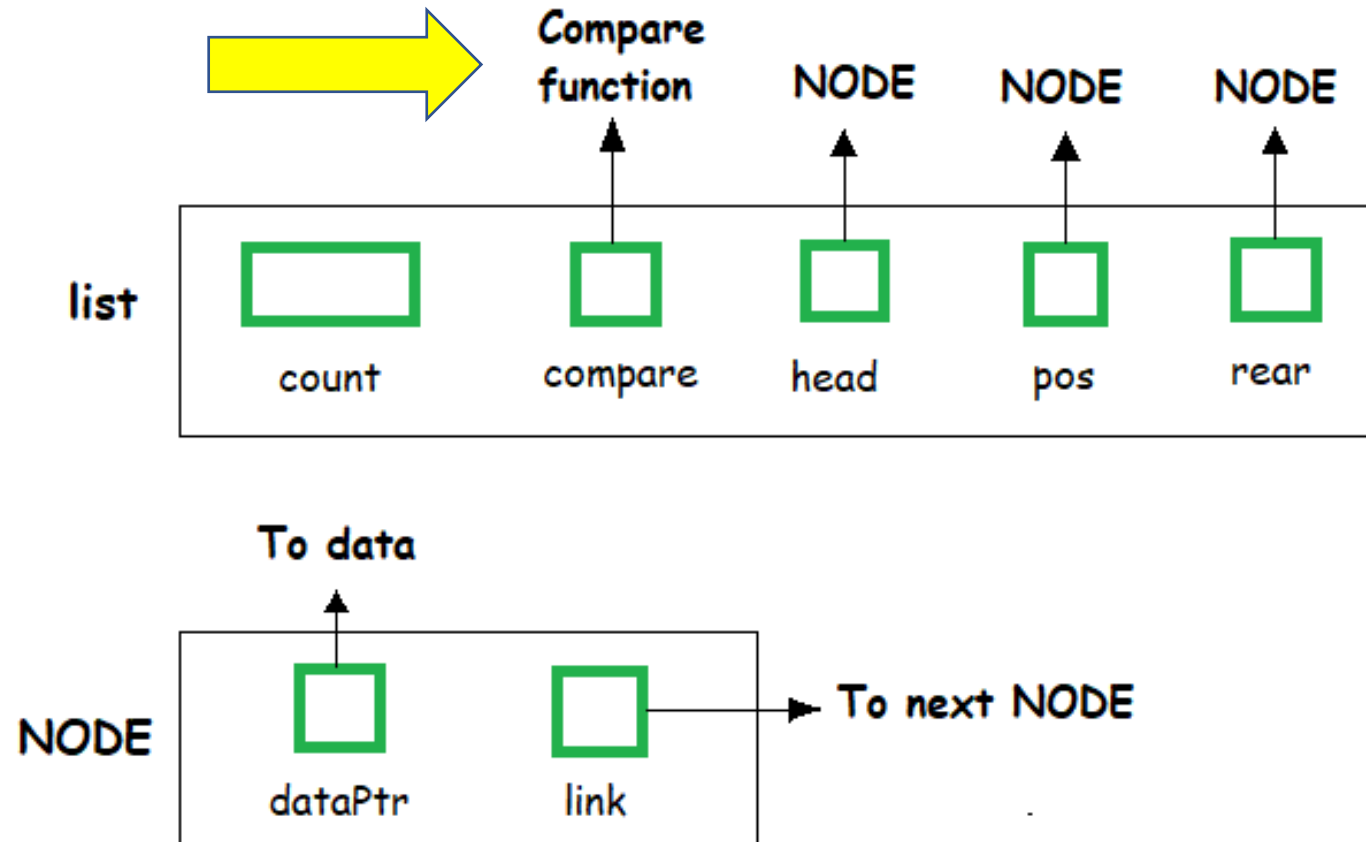
LISTA – Funcionalidades

- Conjunto de **elementos** do **mesmo tipo**
- Armazenados em **ordem sequencial**
- Inserção / remoção / substituição / consulta em **qualquer posição**
- **insert() / remove() / replace() / get()**
- **size() / isEmpty() / isFull()**
- **init() / destroy() / clear()**

O TAD LISTA



O TAD LISTA – Lista ligada de ponteiros



PointersList.h

Pouco habitual utilizar!

```
typedef struct _PointersList List;

List* ListCreate(void);

void ListDestroy(List** p);

void ListClear(List* l);


int ListGetSize(const List* l);

int ListIsEmpty(const List* l);

// Current node functions
int ListGetCurrentIndex(const List* l);

void* ListGetCurrentValue(const List* l);

void ListModifyCurrentValue(const List* l, void* p);
```



PointersList.h

```
// Search
```



```
int ListSearchFromCurrent(const List* l, void* p);
```

```
// Move to functions
```



```
int ListMove(List* l, int newPos);
```

```
int ListMoveToNext(List* l);
```

```
int ListMoveToPrevious(List* l);
```

Seria mais eficiente ter uma lista duplamente ligada!
Onde o da frente tinha um ponteiro para voltar ao anterior

```
int ListMoveToHead(List* l);
```

```
int ListMoveToTail(List* l);
```

PointersList.h


```
// Insert functions ←  
  
void ListInsertBeforeHead(List* l, void* p);  
  
void ListInsertAfterTail(List* l, void* p);  
  
void ListInsertAfterCurrent(List* l, void* p);  
  
void ListInsertBeforeCurrent(List* l, void* p);
```


PointersList.h

```
// Remove functions ←  
  
void ListRemoveHead(List* l);    pop da pilha  
  
void ListRemoveTail(List* l);  
  
void ListRemoveCurrent(List* l);  
  
void ListRemoveNext(List* l);  
  
// Tests ←  
  
void ListTestInvariants(const List* l);
```

PointersList.h

```
struct _PointersListNode {  
    void* data;  
    struct _PointersListNode* next;  
};  
  
struct _PointersList {  
    int size; // current List size  
    struct _PointersListNode* head; // the head of the List  
    struct _PointersListNode* tail; // the tail of the List  
    struct _PointersListNode* current; // the current node  
    int currentPos;  
};
```



Tarefa

- Analisar os ficheiros disponibilizados
- Identificar as **funções incompletas**
- **Implementar** essas funções
- **Testar** com novos exemplos de aplicação



[java2novice.com]

O TAD DEQUE

TAREFA

*** Usar o TAD LISTA como base do TAD DEQUE ***

- Especificar a **interface** do tipo DEQUE, sem qualquer referência ao TAD LISTA – ficheiro .h
- Estabelecer a **representação interna**, usando o TAD **LISTA** – ficheiro .c
- **Implementar** as várias funções, usando as correspondentes **funções** do TAD **LISTA**
- **Testar** com novos exemplos de aplicação