


# Listas Ligadas I

27/10/2025

# 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

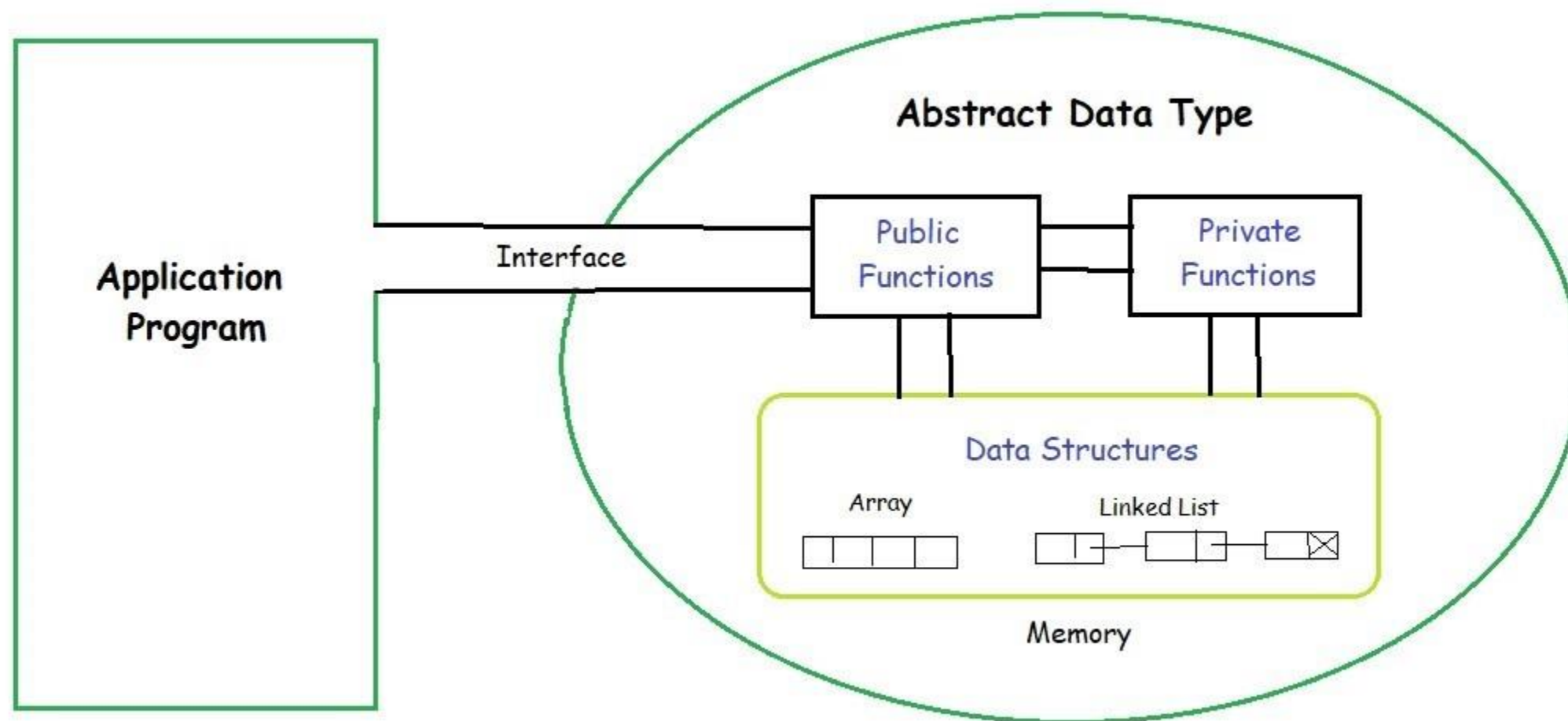
# Sumário

- Recap
- Estruturas de Dados Dinâmicas – Motivação
- O TAD **STACK** – usando uma **lista ligada**
- O TAD **QUEUE** – usando uma **lista ligada**
- O TAD **LISTA LIGADA** – funcionalidades principais
- Exercícios / Tarefas 

Let's  
RECAP

# Recapitulação

# Tipo Abstrato de Dados (TAD)

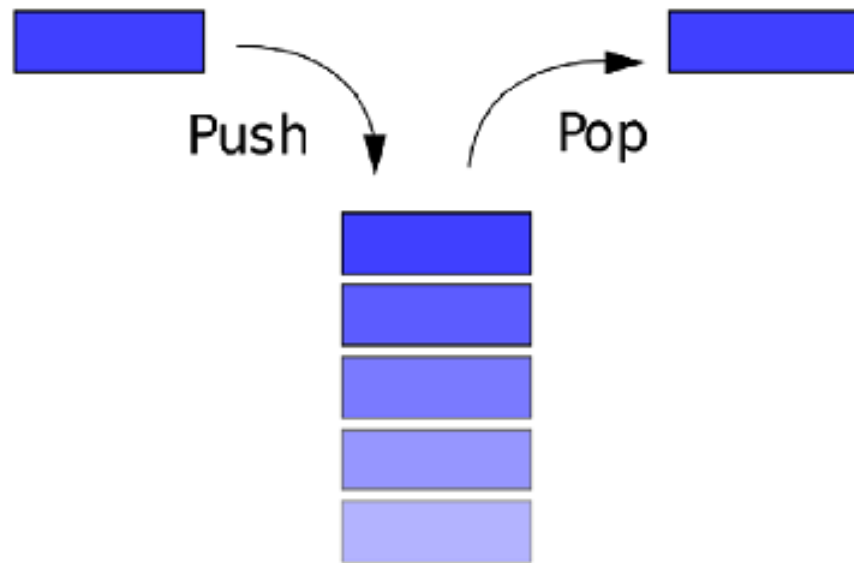


[geeksforgeeks.org]

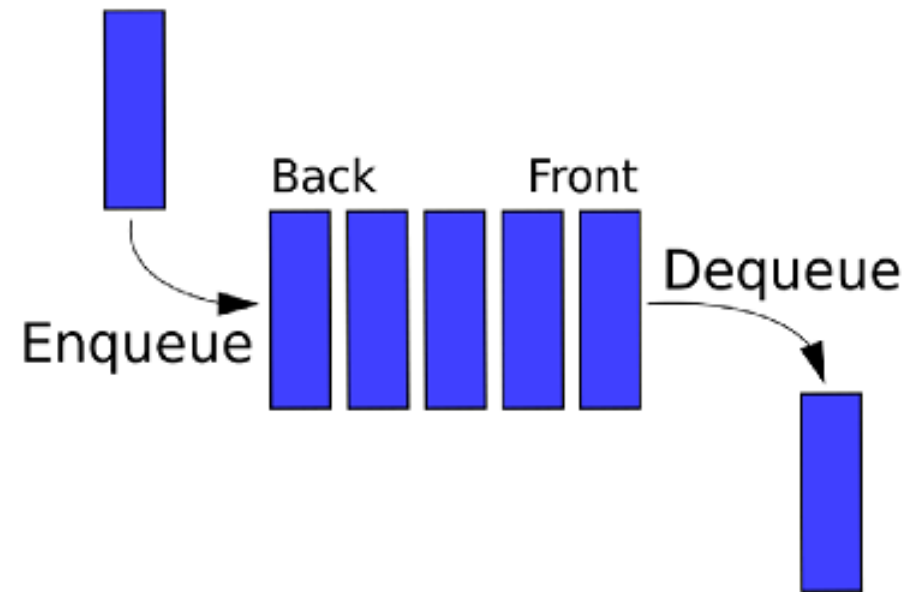
# Tipo Abstrato de Dados (TAD)

- 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



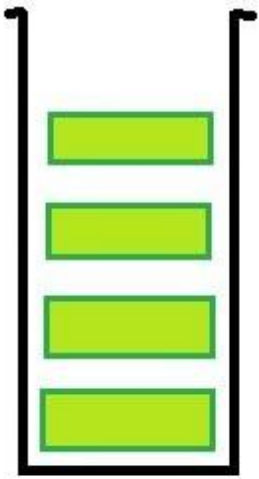
Queue (FIFO) first in first out

[github.io]

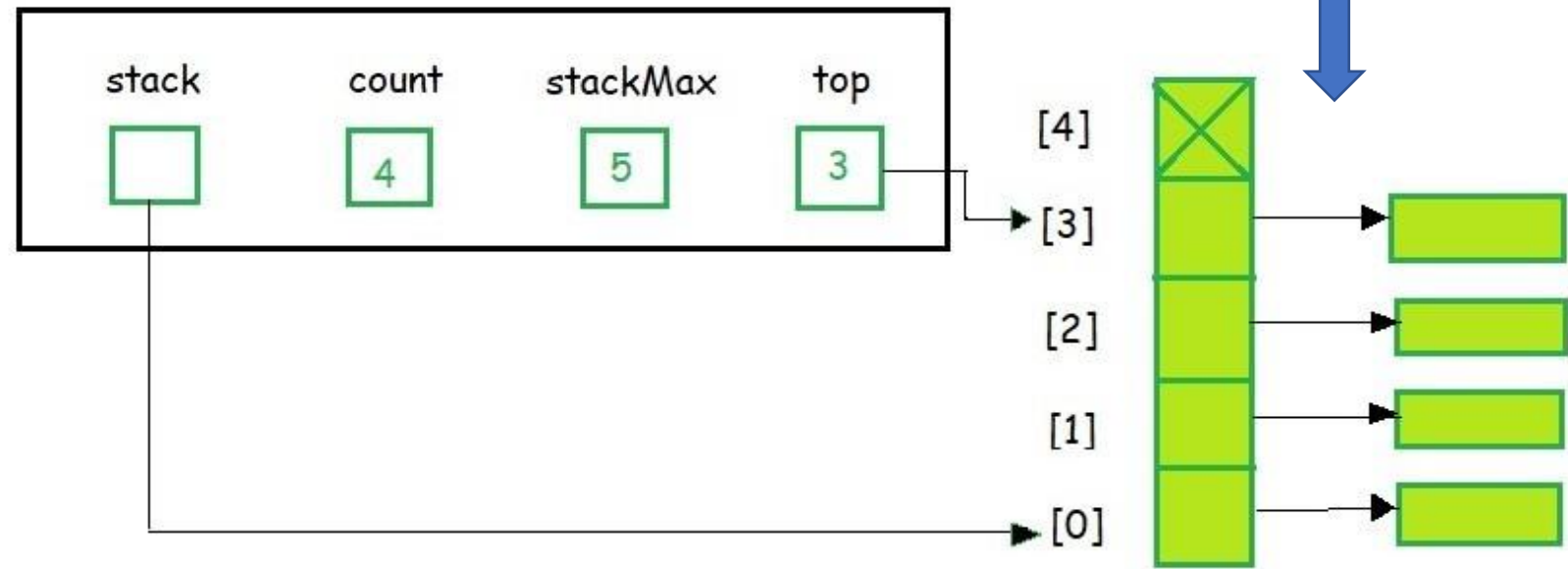
- No próximo **guião prático** iremos usar / aplicar

# O TAD Stack – Array de ponteiros

a) Conceptual



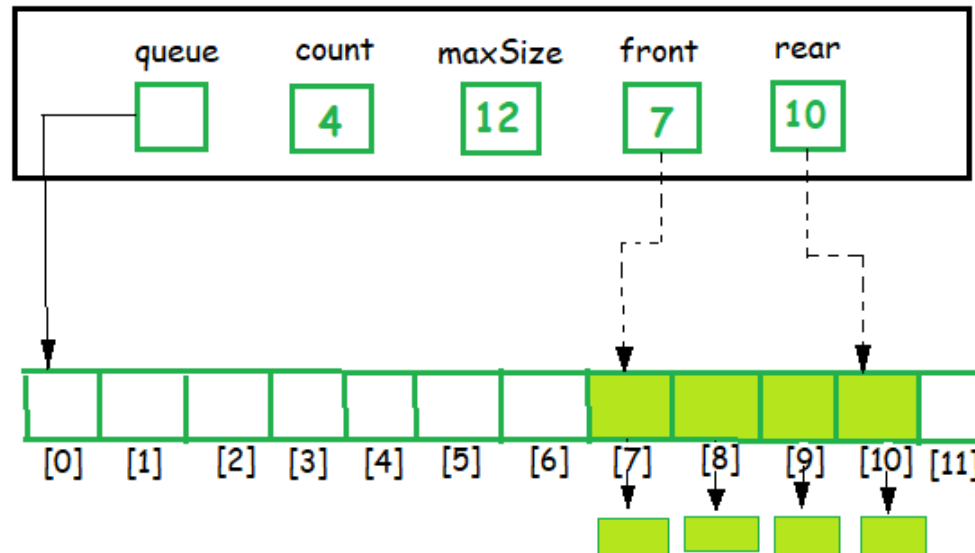
b) Physical Structure



# O TAD QUEUE – Array circular de ponteiros



a) Conceptual



b) Physical Structures

# DEQUE – Tentaram fazer ? – Questões ?



[java2novice.com]

# Estruturas de Dados Dinâmicas

## - Motivação

# Arrays – Estruturas de dados estáticas

- Alocação de um bloco de memória de tamanho fixo
  - Questões em aberto
- O que fazer, quando não sabemos à partida o número de elementos de uma coleção / sequência ?
  - De início, alocar memória em excesso – Desperdício de espaço !!
  - Sempre que necessário, (re)alocar um array de maior tamanho e copiar os elementos para esse novo array – Desperdício de tempo !!
- Como manter a ordem após múltiplas inserções / remoções, sem “espaços vagos” ?
  - Copiar / deslocar elementos para manter a contiguidade – Desperdício de tempo !!

# Listas – Estruturas de dados **dinâmicas**

- **Alternativa**, quando o **número de elementos** é **variável** e não é conhecido antecipadamente
- Escalabilidade e **utilização eficiente da memória**
  - **Armazenamento** e **gestão** de coleções / sequências de **tamanho variável**
  - **Alocação** e **libertação** de espaço de **memória** à medida das necessidades
- **Eficiência** na **inserção** / **remoção** de elementos, mantendo a sua **ordem relativa**
  - Manipulação direta de **ponteiros**

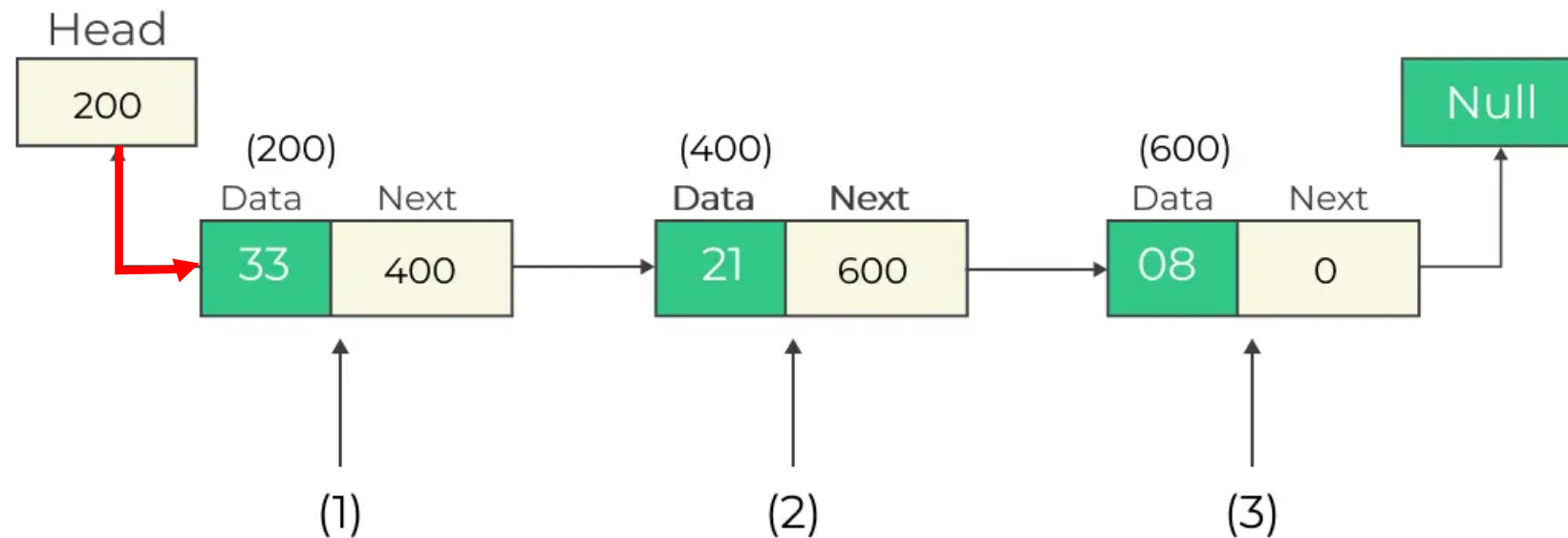
# O TAD STACK / PILHA

## - **Lista** de **Ponteiros** Genéricos



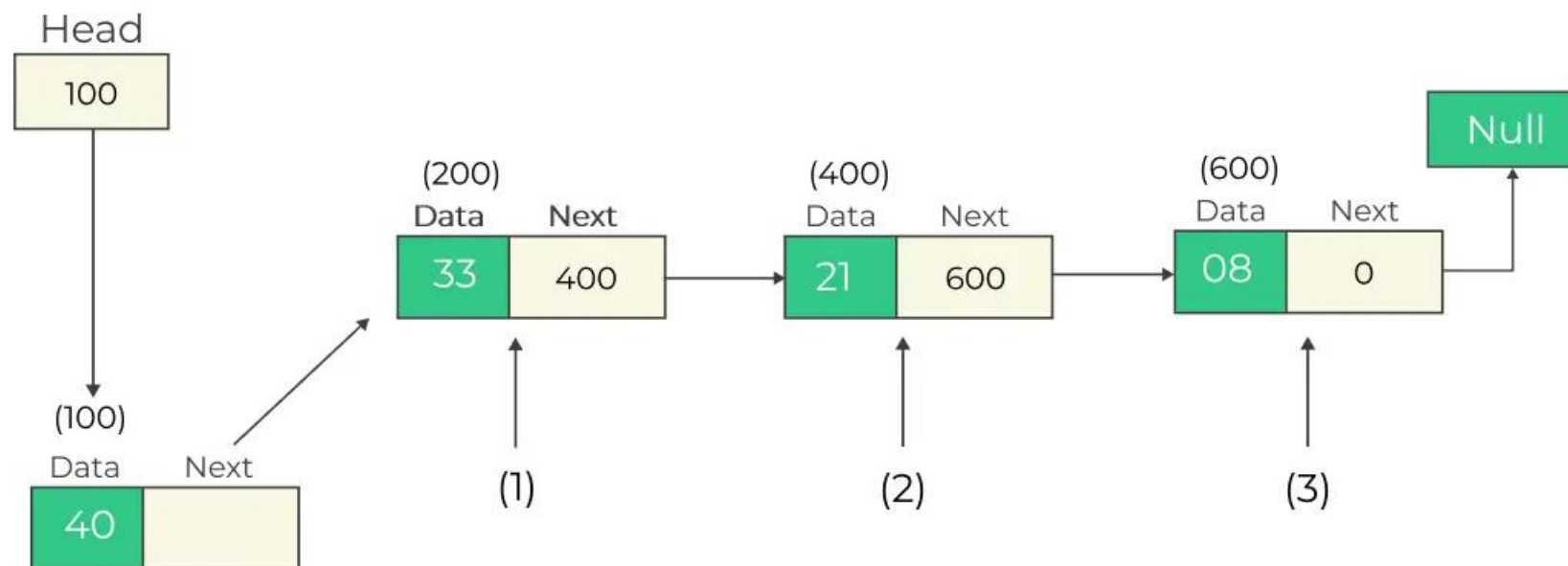
[Wikipedia]

# Pilha de inteiros usando uma **lista ligada**



[prepinsta.com]

# Pilha de inteiros – **push(40)**



[prepinsta.com]

# PointersStack.h

- Sem alterações !
- Alteramos a  
representação interna
- MAS não as  
funcionalidades



```
#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_
```




# PointersStack.c – Nó da Pilha & Cabeçalho

```
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 – Construtor & Destrutor

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



- Alocar o **cabeçalho** da pilha
- A pilha está **vazia**

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

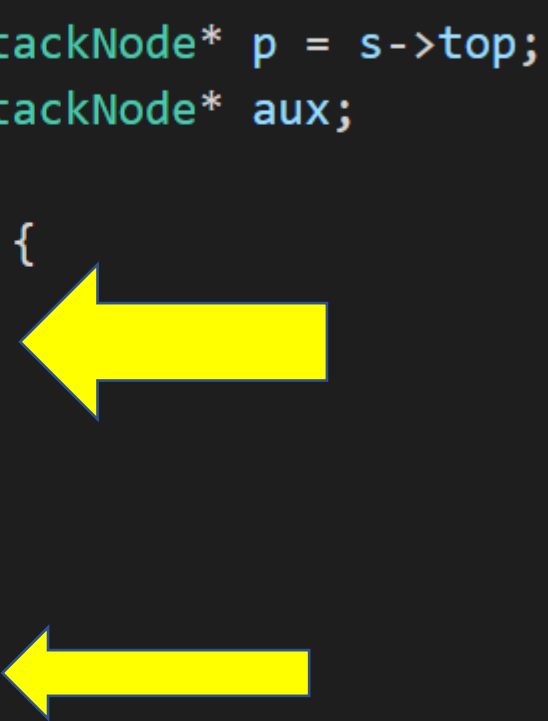


- **Libertar** os **nós** da pilha
- **Libertar** o **cabeçalho**

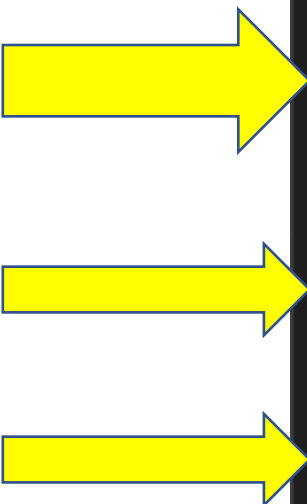
# StackClear() – Eliminar todos os nós, um a um

- 2 ponteiros auxiliares
- Ciclo para percorrer os nós, um a um
- Libertar cada nó, sem perder o ponteiro para o próximo nó, caso exista
- Possível problema : não são libertadas as instâncias referenciadas

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



# StackPush() – Adicionar um nó no início

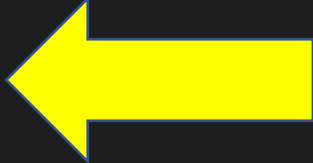



```
void StackPush(Stack* s, void* p) {  
    assert(s != NULL);  
  
    struct _PointersStackNode* aux;  
    aux = (struct _PointersStackNode*)malloc(sizeof(*aux));  
    assert(aux != NULL);  
  
    aux->data = p;  
    aux->next = s->top;  
  
    s->top = aux;  
  
    s->cur_size++;  
}
```

- Alocar o **novo nó**
- Preencher os seus **atributos**
- **Novo topo** da pilha

# StackPop() – Eliminar o 1º nó

```
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;  
}
```



- Retirar o nó do topo da pilha
- Copiar o conteúdo do nó
- Libertar a memória
- Devolver o conteúdo do nó

# TAD PointersStack

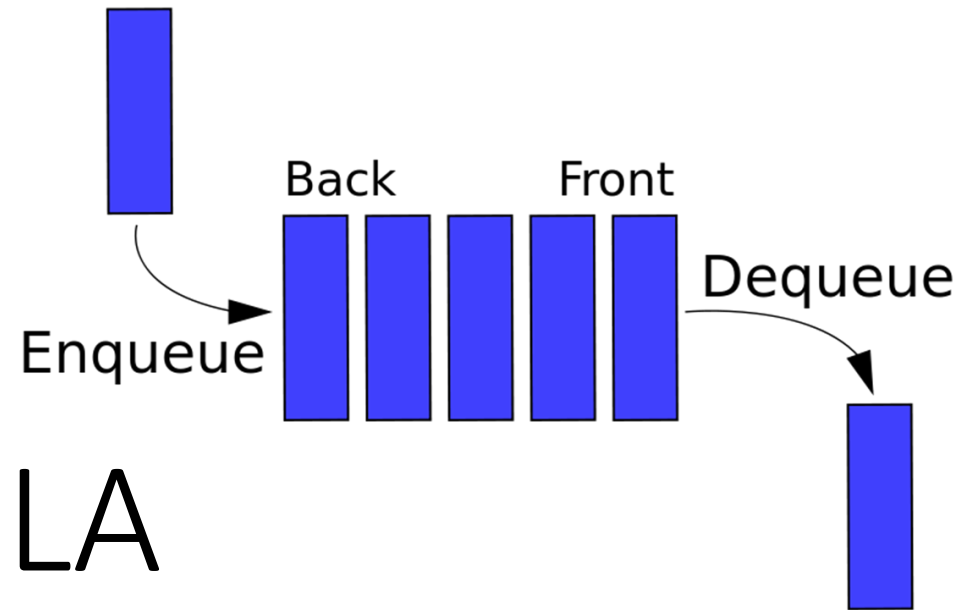


- **TAREFA** : Analisar a **implementação** das **operações** sobre a estrutura de dados **lista ligada** !

# 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

# Fila de inteiros usando uma **lista ligada**

## Adding the elements into Queue



## Removing the elements from Queue



## Printing the Queue



[prepinsta.com]

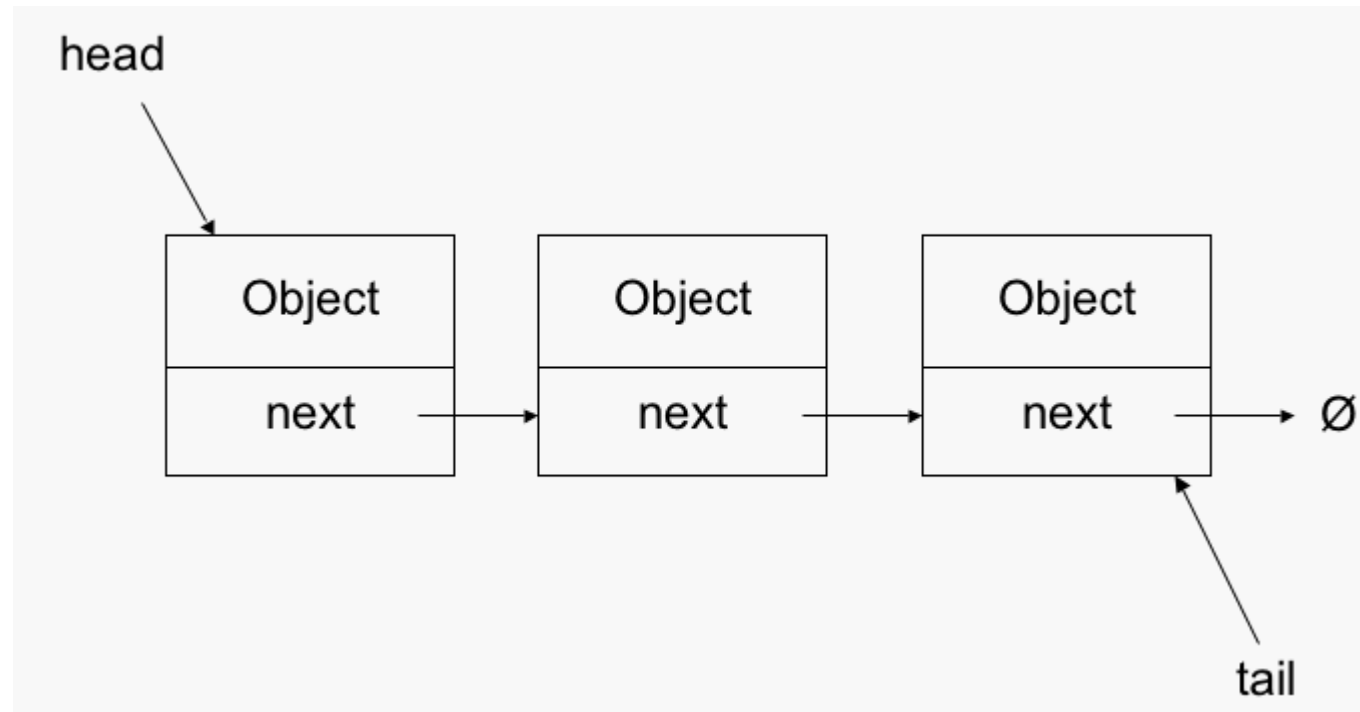
# PointersQueue.h

- Sem alterações !
- Alteramos a **representação interna**
- MAS **não** as **funcionalidades**



```
#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_
```

# Acesso ao início e ao fim da lista ligada




[usfCS]

- Ponteiros para o 1º nó e para o último nó da lista ligada
- Onde é mais fácil inserir um novo nó ? E remover ?

# PointersQueue.c – Nó da Fila & Cabeçalho

```
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 – Construtor & Destrutor

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

- Alocar o **cabeçalho** da fila
- A fila está **vazia**
- **Ponteiros nulos !!**

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

- **Libertar** os **nós** da fila
- **Libertar** o **cabeçalho**

# QueueEnqueue()

- Alocar o novo nó
- Caso a fila só tenha 1 nó, atualizar os 2 ponteiros
- Se não for o caso, atualizar o ponteiro para a cauda da fila

```
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;  
    }  
}
```

# QueueDequeue()

- Retirar o nó da frente da fila
- Caso a fila fique vazia, os 2 ponteiros são nulos
- Se não for o caso, atualizar o ponteiro para a frente da fila
- Libertar a memória
- Devolver o conteúdo do nó

```
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;  
}
```

# TAD PointersQueue

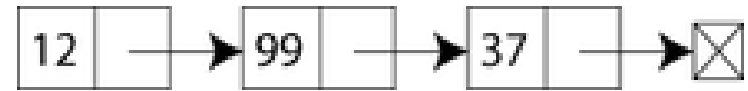


- **TAREFA :** Analisar a **implementação** das **operações** sobre a estrutura de dados **lista ligada !**

Testar o funcionamento do



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



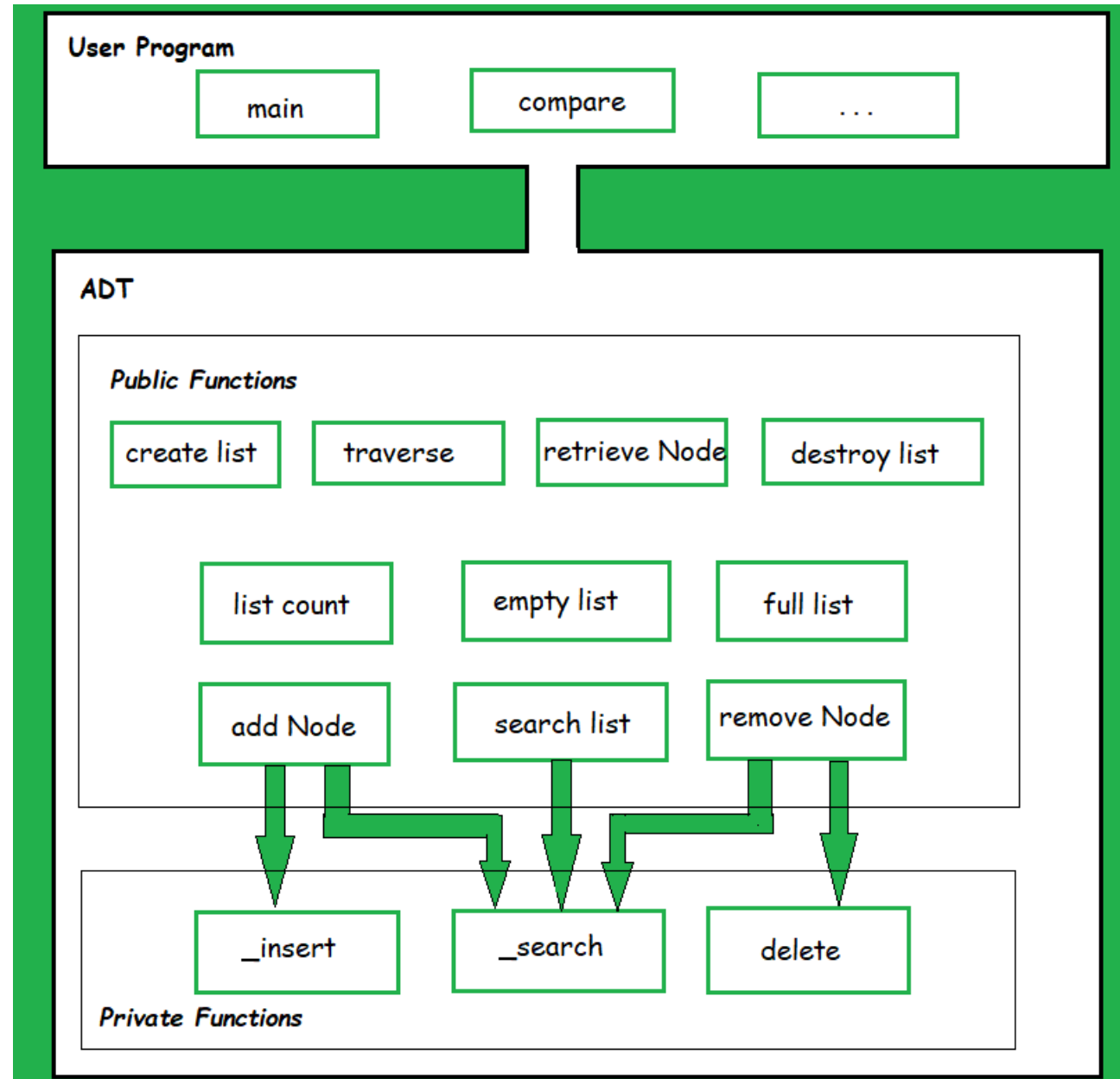
[Wikipedia]

# O TAD **LISTA LIGADA**

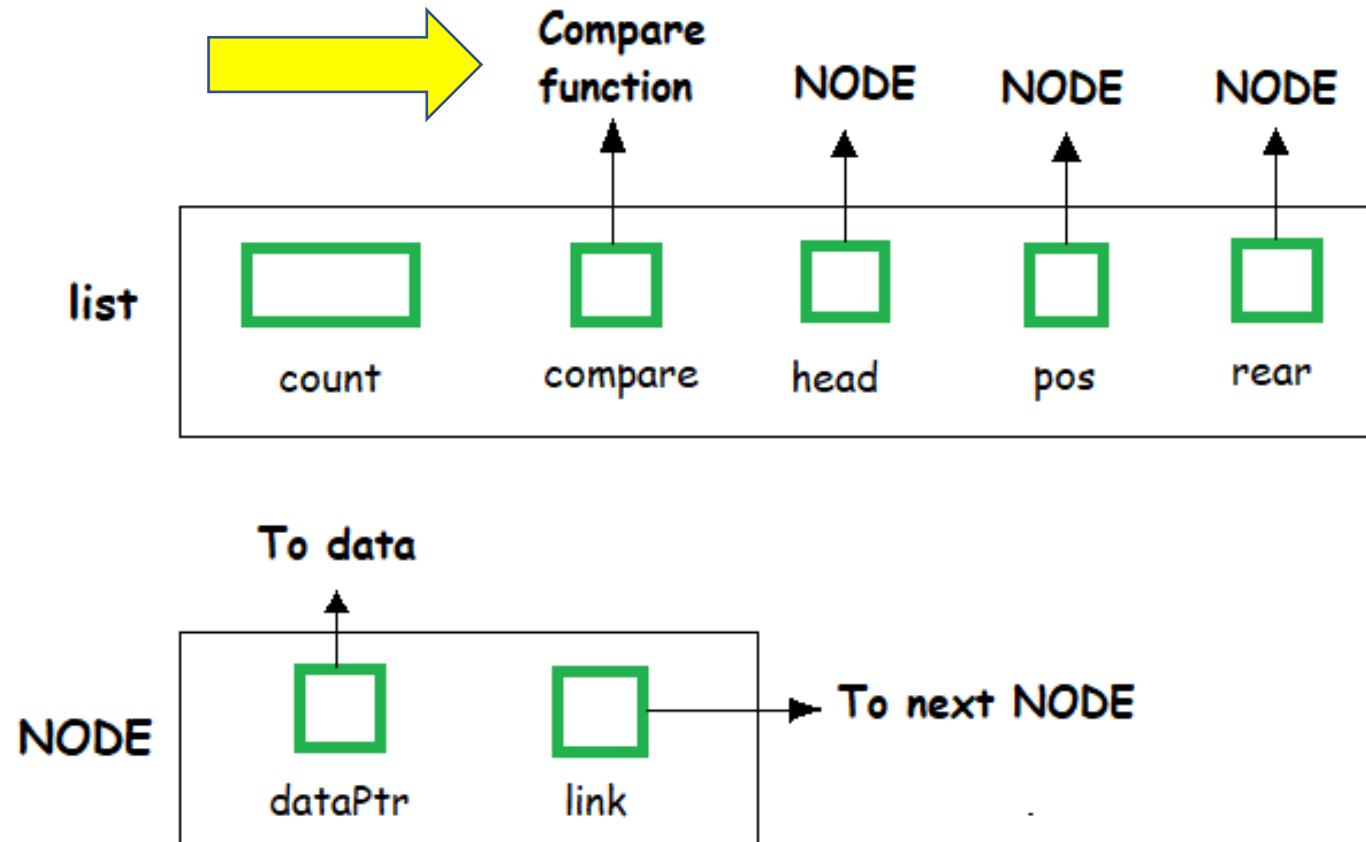
# LISTA – Funcionalidades

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

# O TAD LISTA



# O TAD LISTA – Lista ligada de ponteiros



# PointersList.h – Funcionalidades básicas

```
typedef struct _PointersList List; // Current node functions ←  
  
List* ListCreate(void);  
  
void ListDestroy(List** p);  
  
void ListClear(List* l);  
  
int ListGetSize(const List* l);  
  
int ListIsEmpty(const List* l);  
  
int ListGetCurrentIndex(const List* l);  
  
void* ListGetCurrentValue(const List* l);  
  
void ListModifyCurrentValue(const List* l, void* p);
```

# Funções para iterar sobre a lista

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

# Funções para inserir um novo elemento

```
// 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);
```

# Funções para **remove** um qualquer nó

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

# PointersList.h – Cabeçalho & Nó da lista

```
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** algumas dessas funções
- **Testar** com novos exemplos de aplicação



# Exercícios / Tarefas

# O TAD DEQUE

## - Lista de Ponteiros Genéricos



[java2novice.com]

# TAREFA



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

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