

# Tipos Abstratos de Dados II

Joaquim Madeira

06/05/2021

# Ficheiros com exemplos

- Está disponível no Moodle um **ficheiro ZIP** de suporte aos tópicos de hoje
- Implementação de tipos abstratos usando **diferentes representações** internas
- Exemplos simples de aplicação
- **Implementações incompletas**, que permitem trabalho autónomo de desenvolvimento e teste

# Sumário

- Recap
- O TAD **STACK** – diferentes representações internas
- O TAD **QUEUE** – diferentes representações internas
- O TAD **LIST** – funcionalidades principais
- O TAD **DEQUE** – sugestão adicional

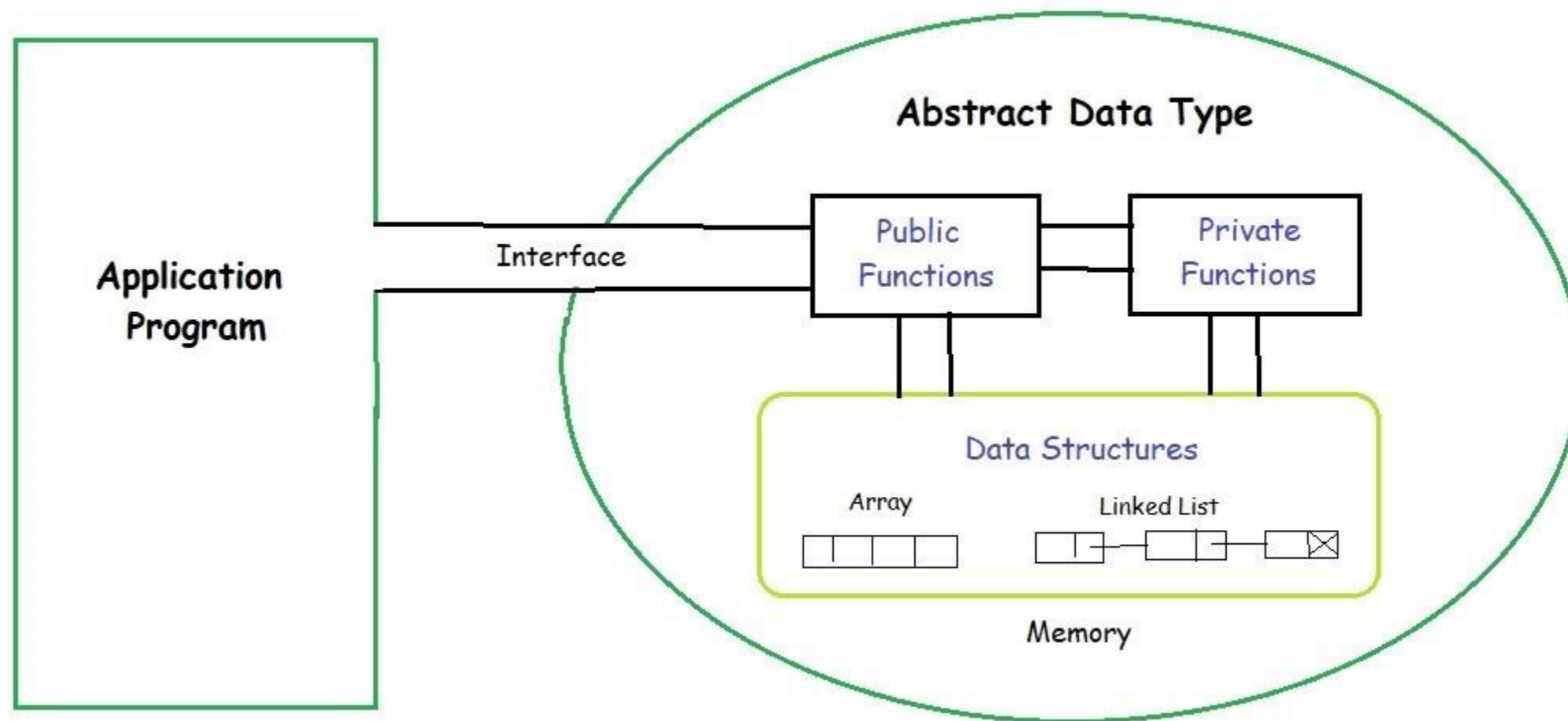
Let's  
RECAP

# Recapitulação

# Motivação

- A linguagem C **não** suporta o paradigma **OO**
- **MAS**, é possível usar alguns princípios de OO no **desenvolvimento** de código em C
- Uma estrutura de dados e as suas operações podem ser organizadas como um **Tipo Abstrato de Dados (TAD)**

# Tipo Abstrato de Dados (TAD)



[geeksforgeeks.org]

# Tipo Abstrato de Dados (TAD)

- Define uma **INTERFACE** entre o TAD e as aplicações que o usam
- **ENCAPSULA** os detalhes da **representação interna** das suas instâncias e da **implementação** das suas funcionalidades
  - Estão **ocultos** para os utilizadores do TAD !!
- Detalhes de representação / implementação **podem ser alterados** sem alterar a interface do TAD
  - **Não** é necessário **alterar código que use o TAD** !!

# Convenções habituais

- O utilizador de um TAD só opera com instâncias através da **interface do TAD**
  - I.e., as suas funções “**públicas**”
- O utilizador está, em geral, **proibido** de aceder diretamente aos campos da representação interna de cada instância
- **Esta convenção também é válida durante os testes do TAD**
  - Os testes avaliam o comportamento de um TAD e não a sua implementação



# Resumo

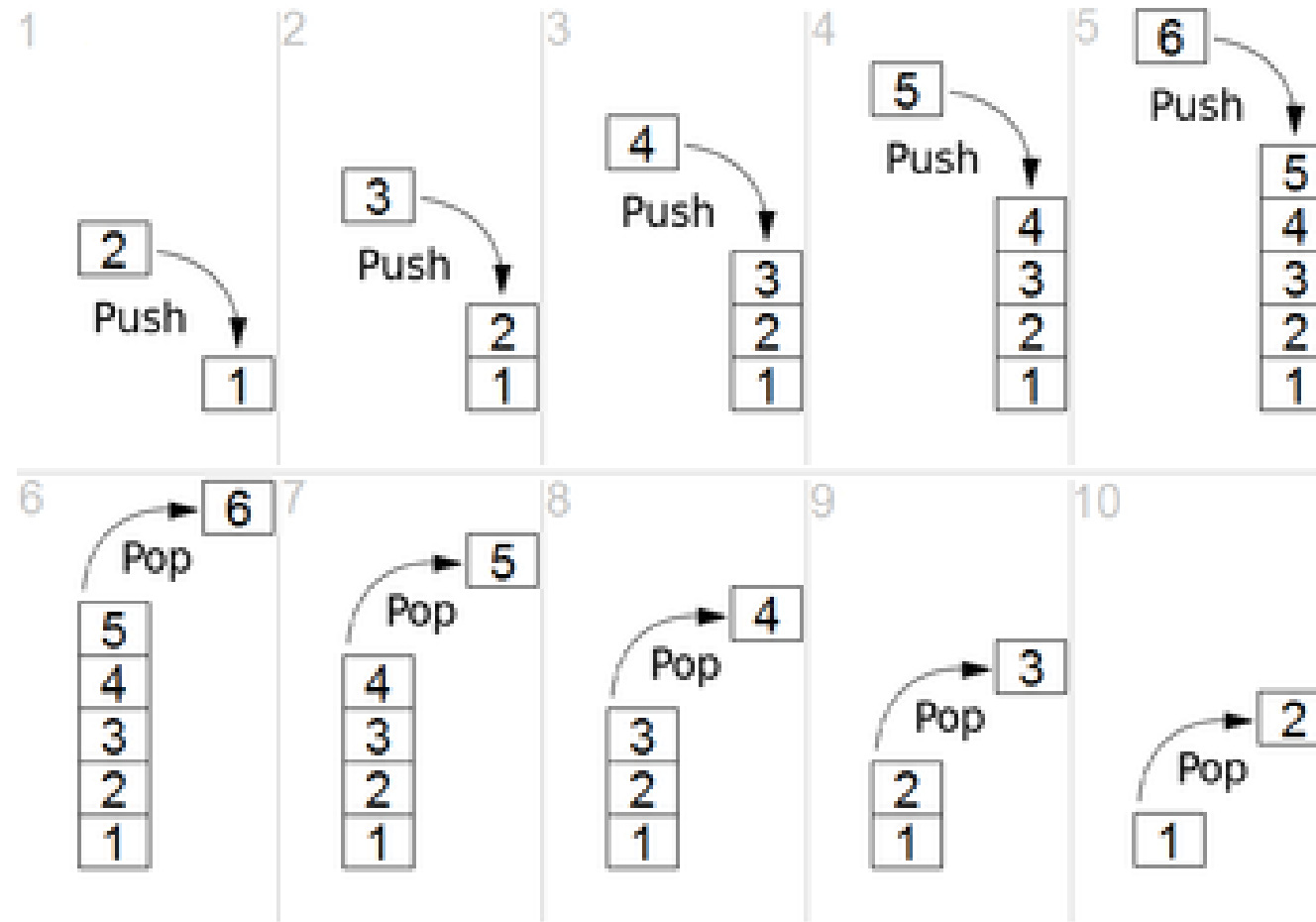
- TAD = especificação + interface + implementação
- Encapsular detalhes da representação / implementação
- Flexibilizar manutenção / reutilização / portabilidade
  
- Ficheiro .h : operações públicas + ponteiro para instância
- Ficheiro .c : implementação + representação interna

# O TAD STACK / PILHA



[Wikipedia]

# STACK / PILHA



[Wikipedia]

# STACK / PILHA – Funcionalidades

- Conjunto de **elementos** do **mesmo tipo**
- Armazenados em **ordem sequencial**
- Inserção / remoção / consulta **apenas** no **topo** da pilha
- **push()** / **pop()** / **peek()**
- **size()** / **isEmpty()** / **isFull()**
- **init()** / **destroy()** / **clear()**

# O TAD STACK / PILHA

## - Array de Inteiros




[Wikipedia]

# IntegersStack.h



```
#ifndef _INTEGERS_STACK_  
#define _INTEGERS_STACK_  
  
typedef struct _IntStack 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);  
  
int StackPeek(const Stack* s);  
  
void StackPush(Stack* s, int i);  
  
int StackPop(Stack* s);  
  
#endif // _INTEGERS_STACK_
```




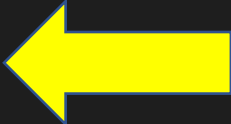
# IntegersStack.c – Array de inteiros

```
#include "IntegersStack.h"   
  
#include <assert.h>  
#include <stdlib.h>  
  
struct _IntStack {  
    int max_size; // maximum stack size  
    int cur_size; // current stack size  
    int* data;    // the stack data (stored in an array)  
};
```

# IntegersStack.c



```
Stack* StackCreate(int size) {  
    assert(size >= 10 && size <= 1000000);  
    Stack* s = (Stack*)malloc(sizeof(Stack));  
    if (s == NULL) return NULL;  
    s->max_size = size;  
    s->cur_size = 0;  
    s->data = (int*)malloc(size * sizeof(int));  
    if (s->data == NULL) {  
        free(s);  
        return NULL;  
    }  
    return s;  
}
```



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



# IntegersStack.c

```
void StackClear(Stack* s) { s->cur_size = 0; }

int StackSize(const Stack* s) { return s->cur_size; }

int StackIsFull(const Stack* s) { return (s->cur_size == s->max_size) ? 1 : 0; }

int StackIsEmpty(const Stack* s) { return (s->cur_size == 0) ? 1 : 0; }
```

# IntegersStack.c

```
int StackPeek(const Stack* s) {
    assert(s->cur_size > 0);
    return s->data[s->cur_size - 1];
}

void StackPush(Stack* s, int i) {
    assert(s->cur_size < s->max_size);
    s->data[s->cur_size++] = i;
}

int StackPop(Stack* s) {
    assert(s->cur_size > 0);
    return s->data[--(s->cur_size)];
}
```

# Aplicação – Escrever pela ordem inversa

- Como escrever pela **ordem inversa** os **algarismos** de um **número** inteiro positivo ?
- Como se pode utilizar o **TAD STACK** ?
- **TAREFA** : Analisar o exemplo de aplicação !!

# O TAD STACK / PILHA

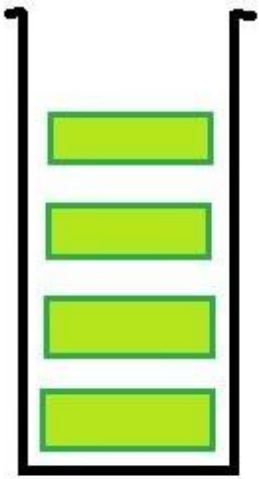
- *Array* de *Ponteiros* Genéricos



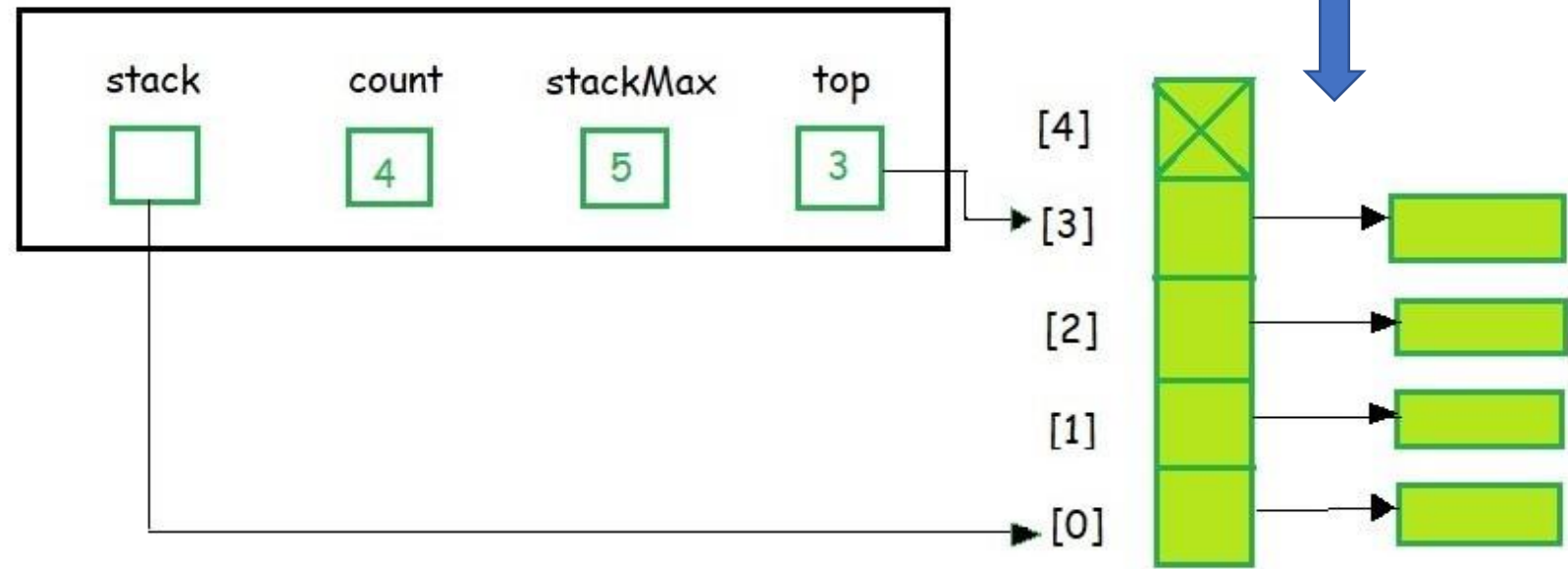
[Wikipedia]

# O TAD Stack – Array de ponteiros

a) Conceptual



b) Physical Structure



# PointersStack.h

- Alterações ?

```
#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_
```

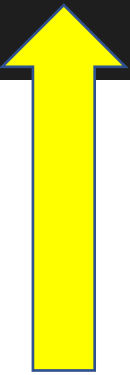
# PointersStack.h

- Alterações ?

```
#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

```
struct _PointersStack {  
    int max_size; // maximum stack size  
    int cur_size; // current stack size  
    void** data;  // the stack data (pointers stored in an array)  
};
```





# PointersStack.c

- **TAREFA** : Analisar a **implementação** das funções do TAD
- Quais são as diferenças ?

# Aplicação – Escrever pela ordem inversa

- Já sabemos como escrever pela **ordem inversa** os **algarismos** de um **número** inteiro positivo
- Como se pode utilizar esta **nova versão** do **TAD STACK** ?
- Qual é a **principal diferença** ?
- **TAREFA** : Analisar o exemplo de aplicação !!

# O TAD STACK / PILHA

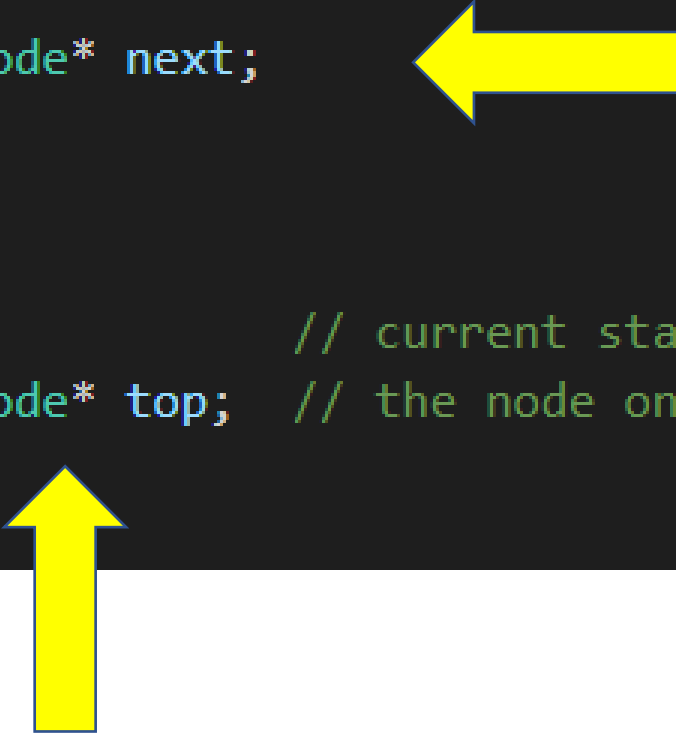
## - Lista de Ponteiros Genéricos



[Wikipedia]


# O TAD Stack – Lista ligada de ponteiros

```
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

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

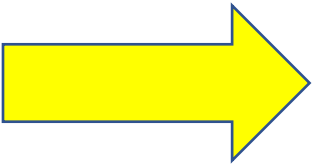


```
void StackDestroy(Stack** p) {  
    assert(*p != NULL);  
    Stack* s = *p;  
  
    StackClear(s);  
  
    free(s);  
    *p = NULL;  
}
```



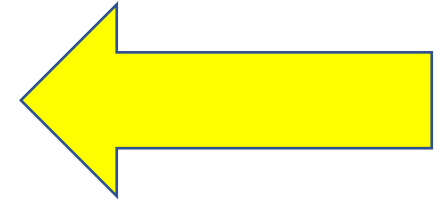
# PointersStack.c

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



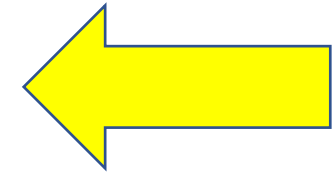
# PointersStack.c

```
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++;  
}
```



# 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;  
}
```



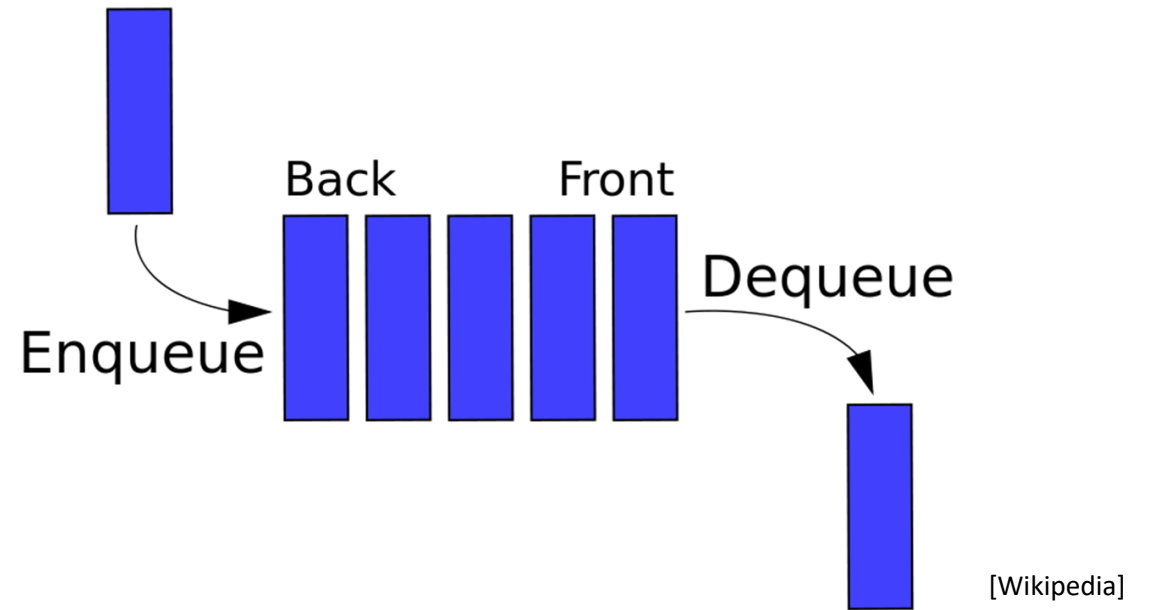


# PointersStack.h + PointersStack.c

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

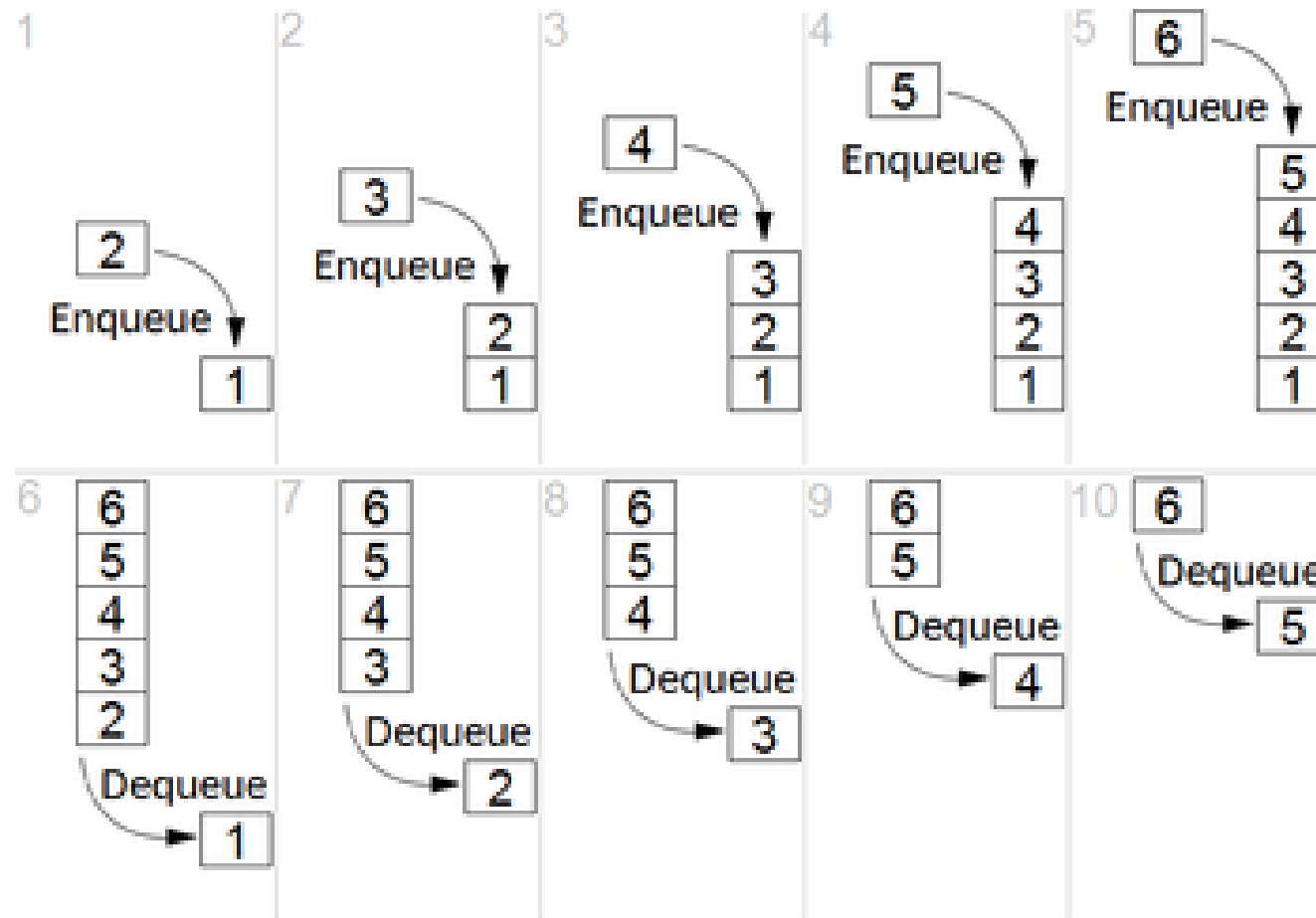
# Aplicação – Escrever pela ordem inversa

- 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 !!



# O TAD QUEUE / FILA

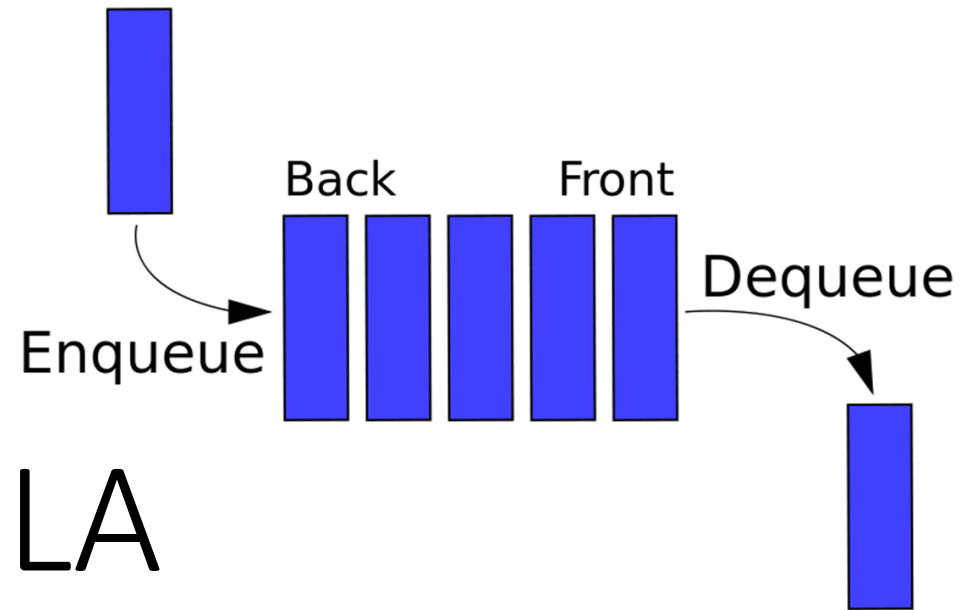
# QUEUE / FILA



[Wikipedia]

# QUEUE / FILA – Funcionalidades

- Conjunto de **elementos** do **mesmo tipo**
- Armazenados em **ordem sequencial**
- Inserção na **cauda** da fila
- Remoção / consulta **apenas** na **frente** da fila
- **enqueue() / dequeue() / peek()**
- **size() / isEmpty() / isFull()**
- **init() / destroy() / clear()**



[Wikipedia]

# O TAD QUEUE / FILA

- Lista de Ponteiros Genéricos

# 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);

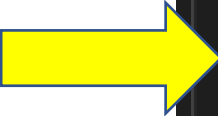
void QueueEnqueue(Queue* q, void* p);

void* QueueDequeue(Queue* q);

#endif // _POINTERS_QUEUE_
```

# O TAD QUEUE – Lista ligada de ponteiros

```
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  
};
```



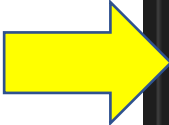


# PointersQueue.c

```
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



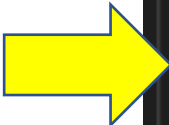
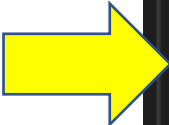
```
void QueueEnqueue(Queue* q, void* p) {
    assert(q != NULL);

    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;
    }
}
```



# PointersQueue.c

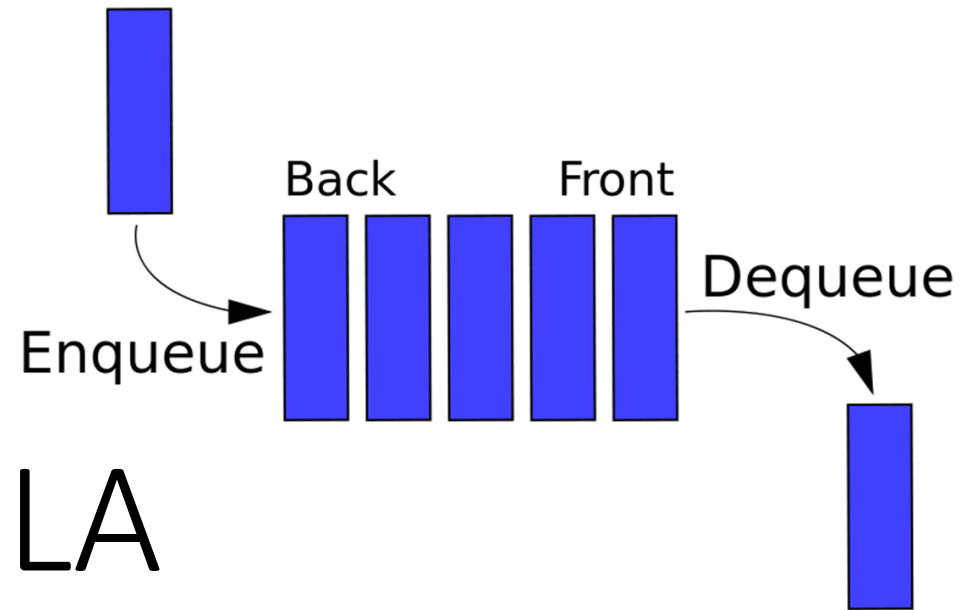
```
void* QueueDequeue(Queue* q) {  
    assert(q != NULL && q->size > 0);  
  
    struct _PointersQueueNode* aux = q->head;  
    void* p = aux->data;  
  
    q->size--;  
  
    if (q->size == 0) {  
        q->head = NULL;  
        q->tail = NULL;  
    } else {  
        q->head = aux->next; 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

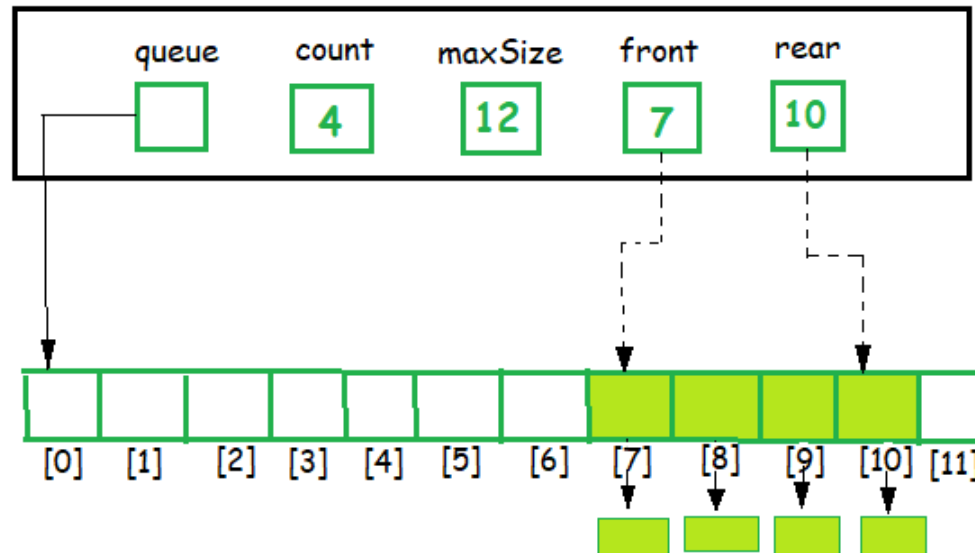
# O TAD QUEUE / FILA

- Array Circular de Ponteiros

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



a) Conceptual



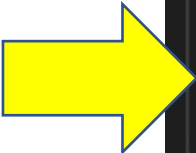
b) Physical Structures

# PointersQueue.c

```
struct _PointersQueue {  
    int max_size; // maximum Queue size  
    int cur_size; // current Queue size  
    int head;  
    int tail;  
    void** data; // the Queue data (pointers stored in an array)  
};  
  
// PRIVATE auxiliary function  
  
static int increment_index(const Queue* q, int i) {  
    return (i + 1 < q->max_size) ? i + 1 : 0;  
}
```



# PointersQueue.c



```
Queue* QueueCreate(int size) {
    assert(size >= 10 && size <= 1000000);
    Queue* q = (Queue*)malloc(sizeof(Queue));
    if (q == NULL) return NULL;


    q->max_size = size;
    q->cur_size = 0;

    q->head = 1; // cur_size = tail - head + 1
    q->tail = 0;

    q->data = (int*)malloc(size * sizeof(int));
    if (q->data == NULL) {
        free(q);
        return NULL;
    }
    return q;
}
```

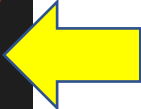
```
void QueueDestroy(Queue** p) {
    assert(*p != NULL);
    Queue* q = *p;
    free(q->data);
    free(q);
    *p = NULL;
}
```

# PointersQueue.c



```
void QueueEnqueue(Queue* q, int i) {  
    assert(q->cur_size < q->max_size);  
    q->tail = increment_index(q, q->tail);  
    q->data[q->tail] = i;  
    q->cur_size++;  
}
```

```
int QueueDequeue(Queue* q) {  
    assert(q->cur_size > 0);  
    int old_head = q->head;  
    q->head = increment_index(q, q->head);  
    q->cur_size--;  
    return q->data[old_head];  
}
```

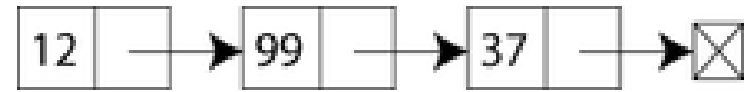


# PointersQueue.h + PointersQueue.c

- **TAREFA** : Analisar a **implementação** das funções do TAD
- Analisar / Estudar a implementação das **operações** sobre o **array circular** !

# Aplicação – Testar o funcionamento do TAD

- São necessárias **modificações** no código do exemplo para se utilizar esta **nova versão** do **TAD STACK** ?
- **TAREFA** : Analisar o exemplo de aplicação !!



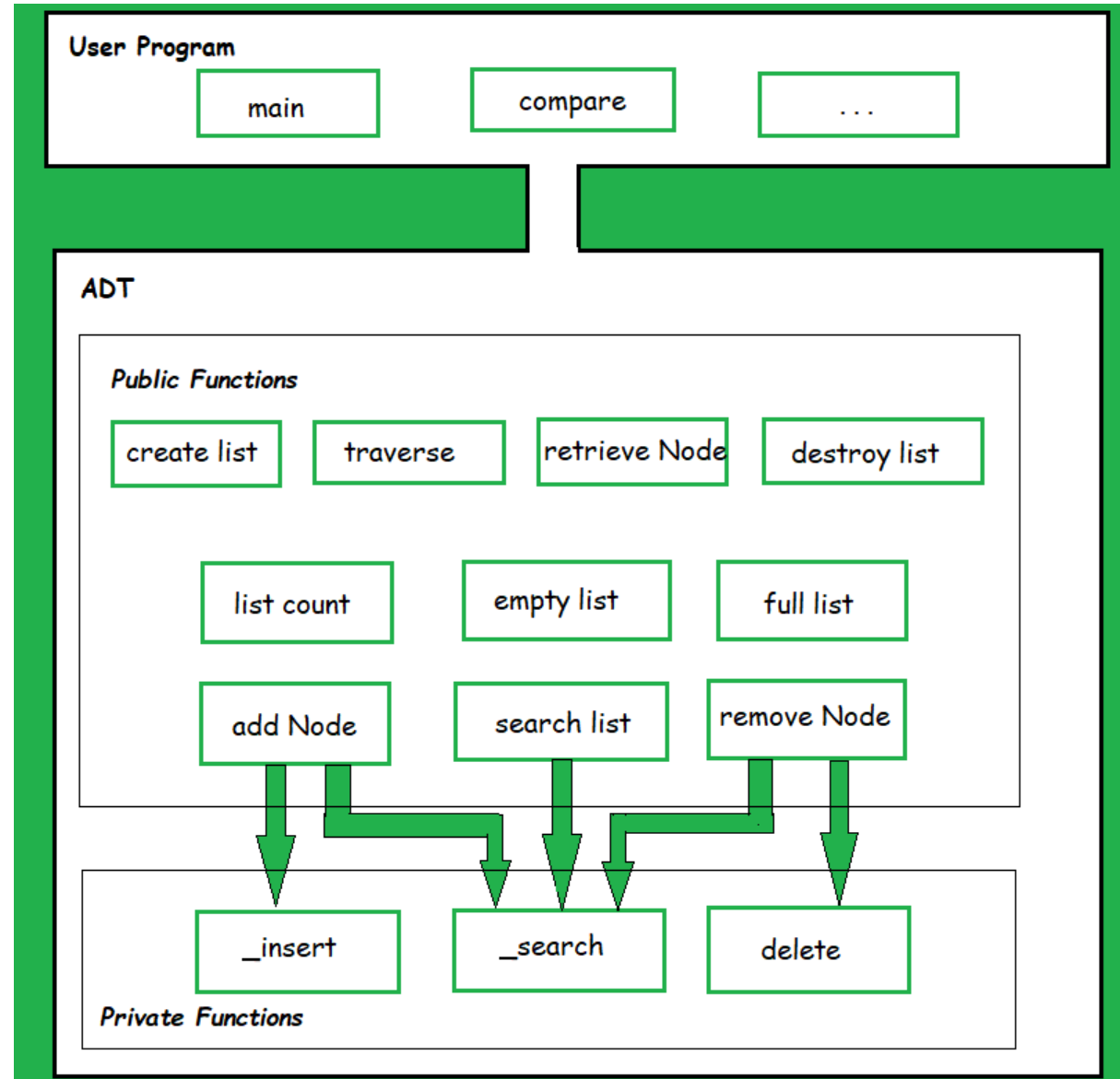
[Wikipedia]

# O TAD LISTA

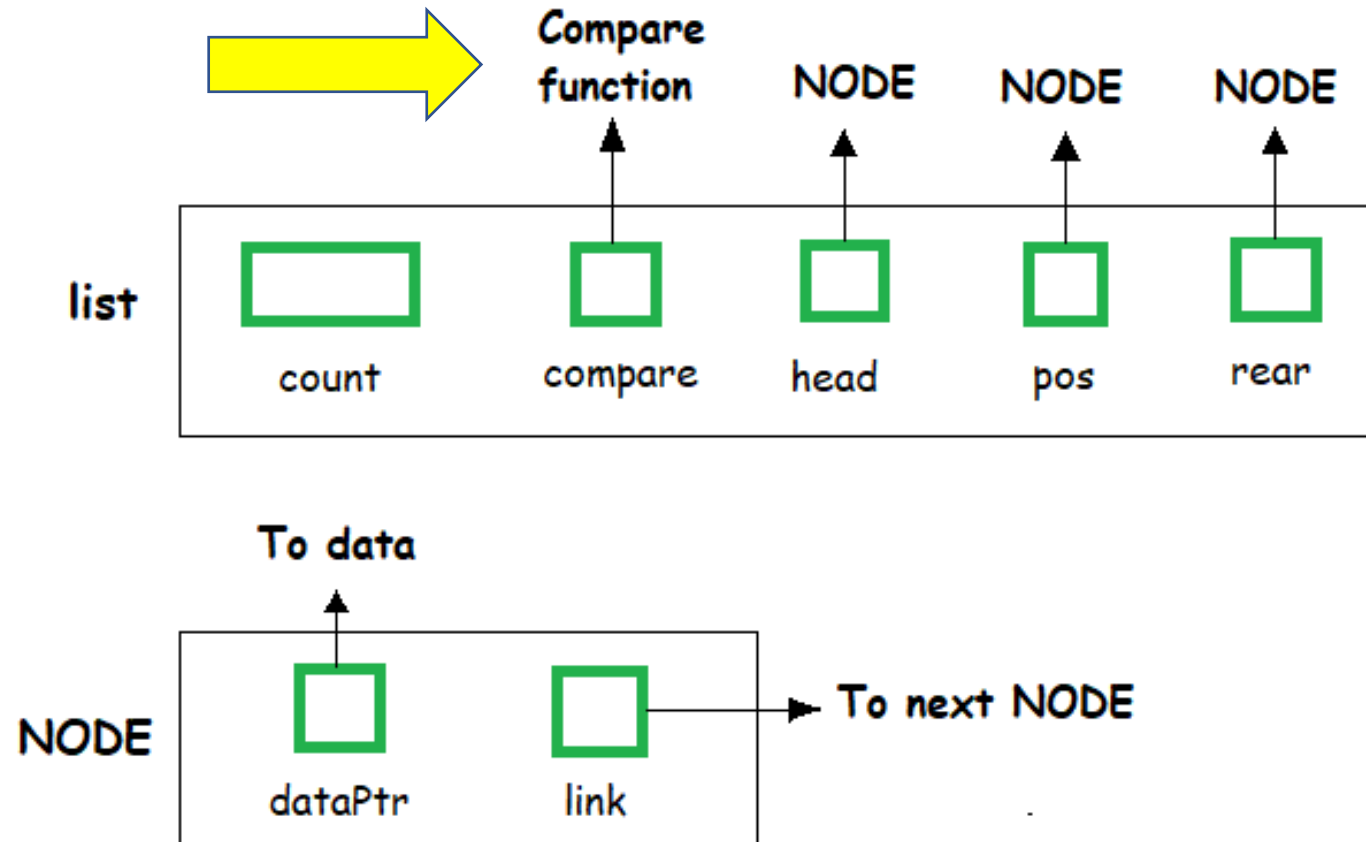
# LISTA – Funcionalidades

- 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

```
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);
```

```
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);  
  
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

- Especificar a **interface** do tipo DEQUE – ficheiro **.h**
- Estabelecer a **representação interna**, usando um **ARRAY CIRCULAR** – ficheiro **.c**
- **Implementar** as várias funções
- **Testar** com novos exemplos de aplicação
- **Sugestão:** atenda às semelhanças com o TAD QUEUE implementado com um array circular



# 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