

Tipos Abstratos de Dados II

Joaquim Madeira

23/04/2020

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
- O TAD DEQUE – sugestão adicional

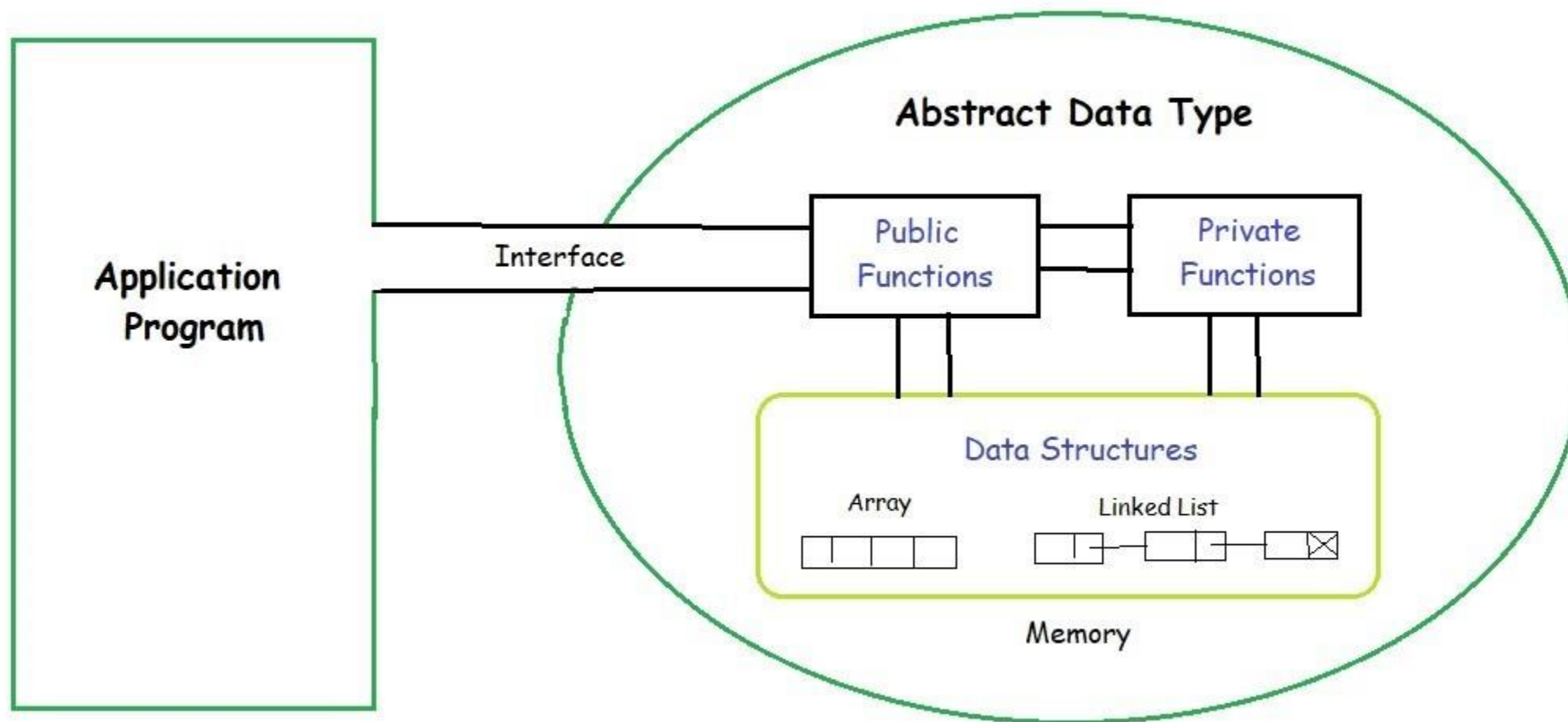
Let's
RECAP

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

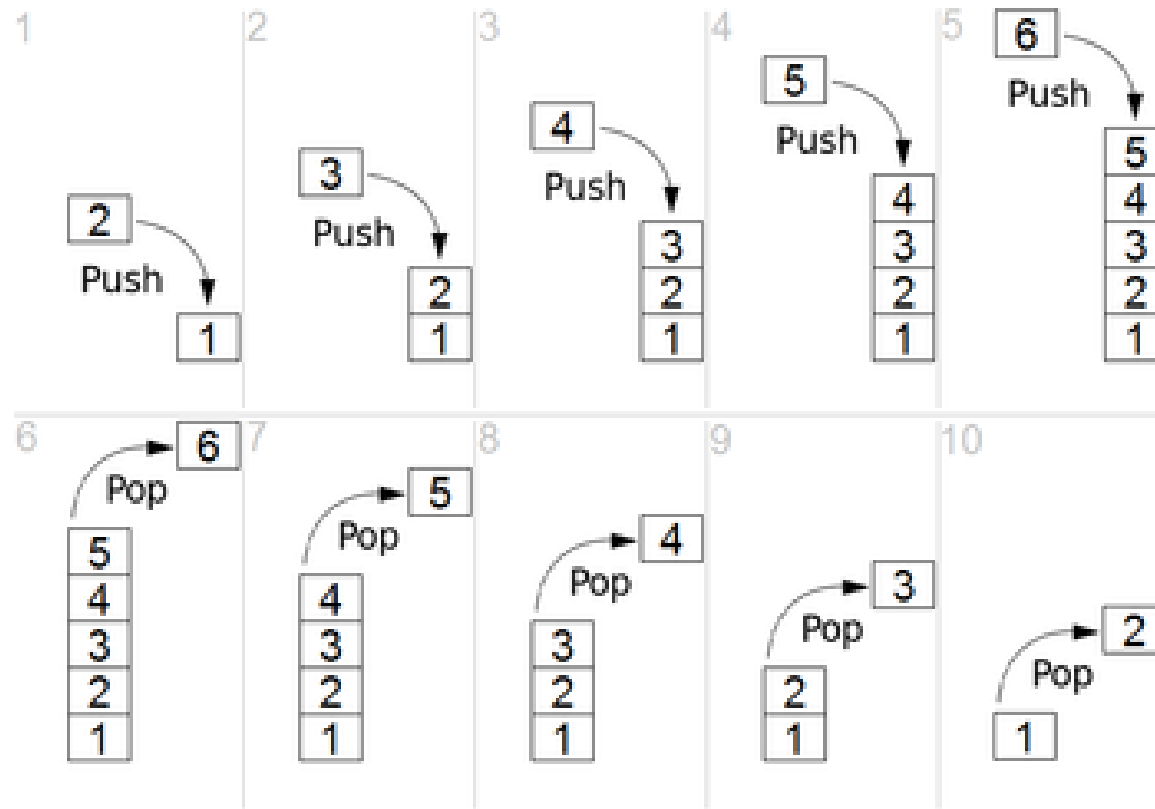
- Ficheiro .h : operações públicas + ponteiro para instância
- Ficheiro .c : implementação + representação interna

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()**

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 – Representação com **array**

```
#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

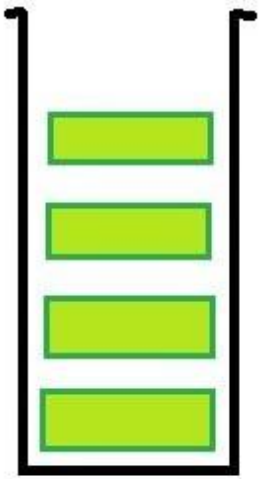
- Vamos **abrir o ficheiro** e **analisar** a implementação de algumas funções

Aplicação – Escrever pela ordem inversa

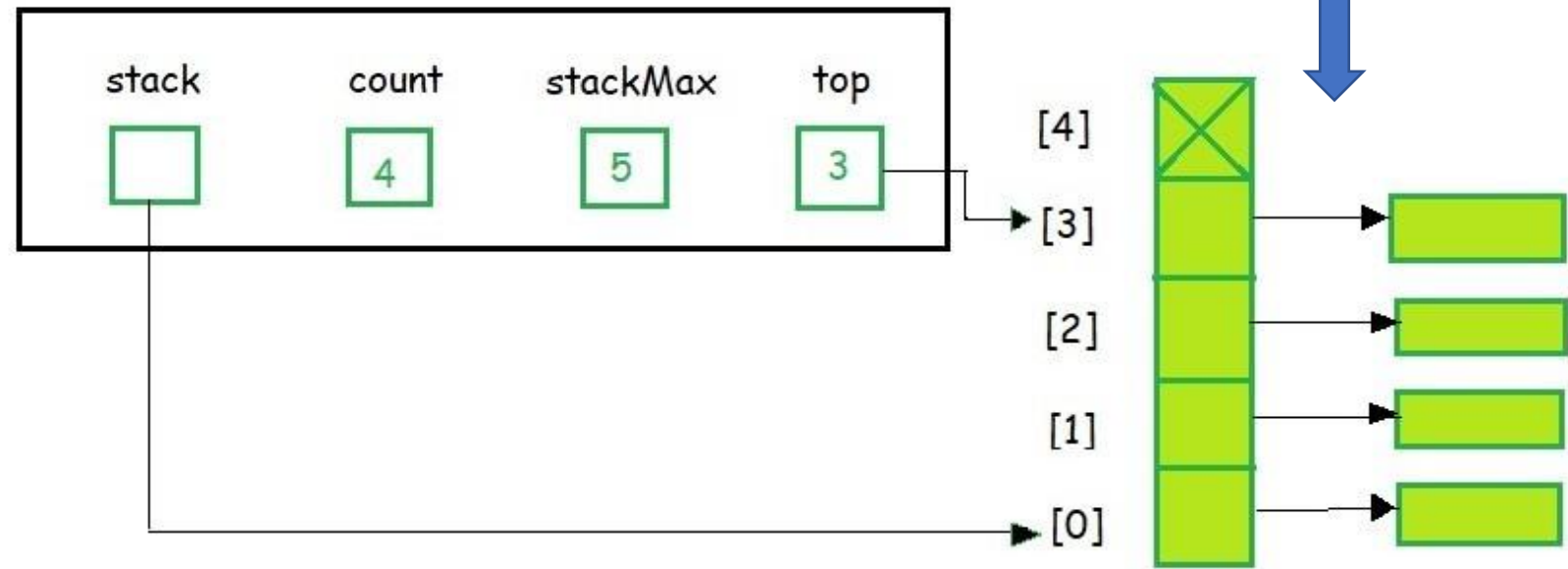
- Como escrever pela **ordem inversa** os **algarismos** de um **número** inteiro positivo ?
- Como se pode utilizar o **TAD STACK** ?
- Analisar o exemplo de aplicação

O TAD Stack – Array de ponteiros

a) Conceptual



b) Physical Structure



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);

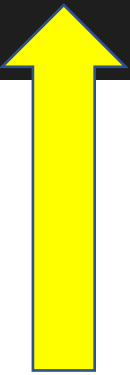
void StackPush(Stack* s, void* p);

void* StackPop(Stack* s);

#endif // _POINTERS_STACK_
```

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)  
};
```

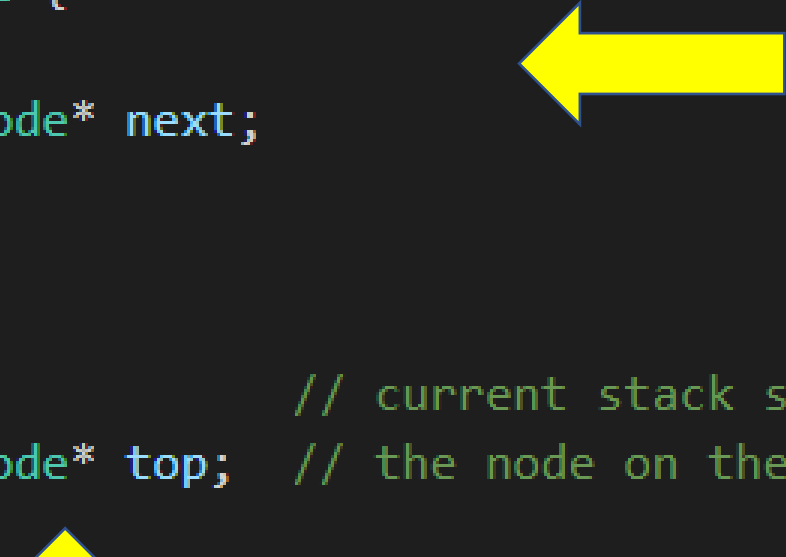


PointersStack.c

- Vamos **abrir o ficheiro** e **analisar** a implementação de algumas funções
- Quais são as diferenças ?

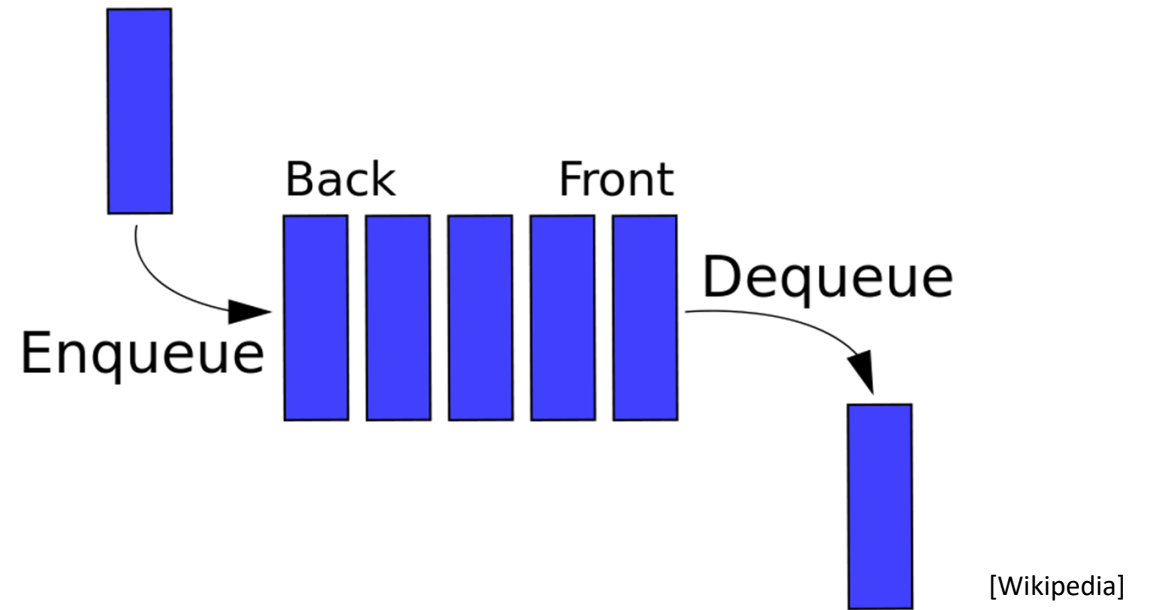
O TAD Stack – Lista ligada de ponteiros

```
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  
};
```



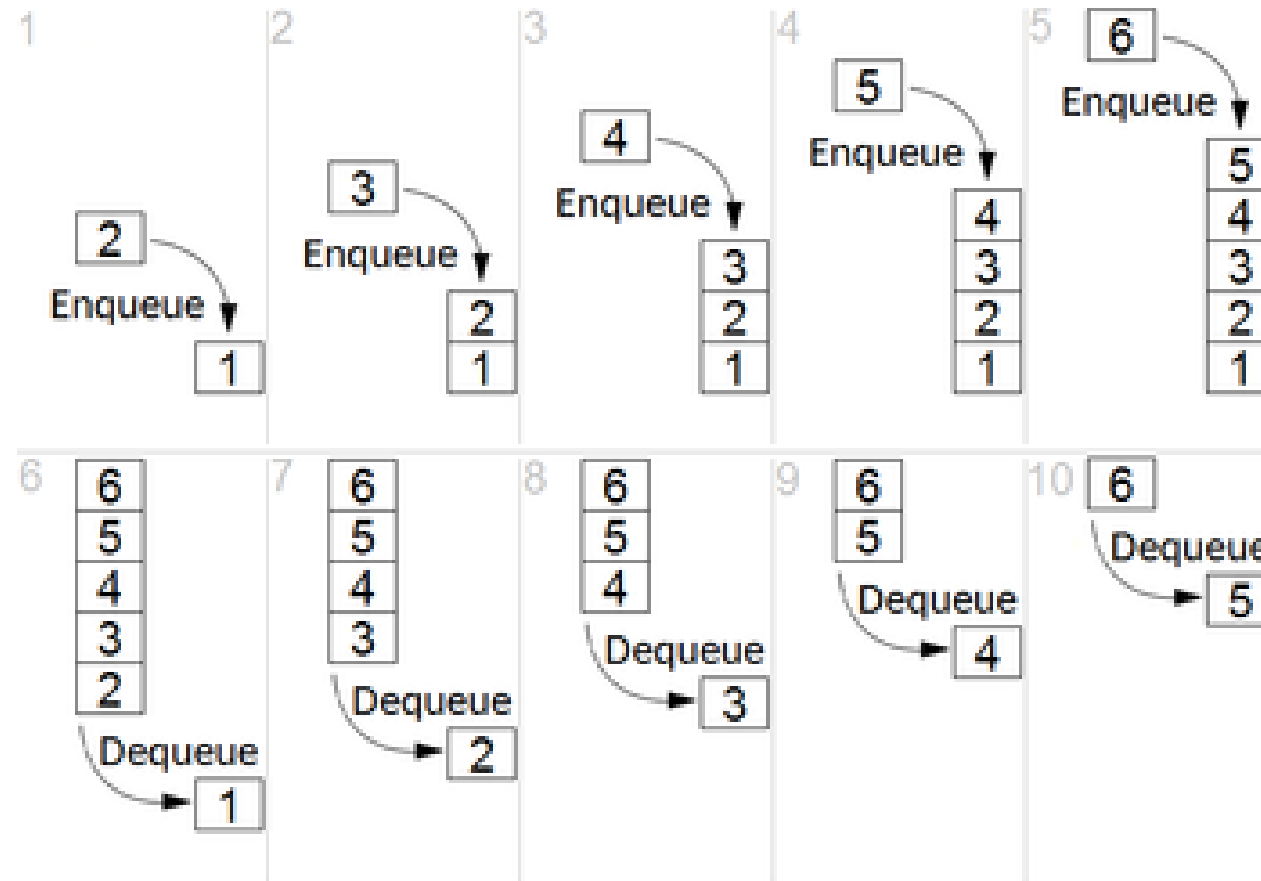
PointersStack.h + PointersStack.c

- Vamos **abrir os ficheiros** e **analisar** a implementação de algumas funções
- Quais são as diferenças ?



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()**

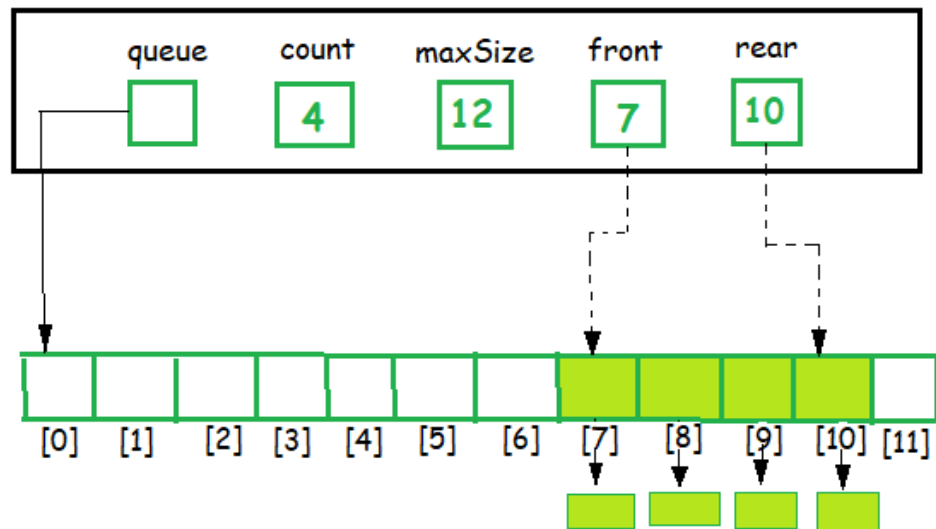
PointersQueue.h

```
#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

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

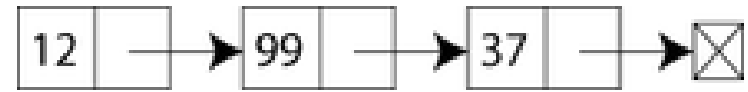
- Vamos **abrir o ficheiro** e **analisar** a implementação de algumas funções

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

- Vamos **abrir o ficheiro** e **analisar** a implementação de algumas funções



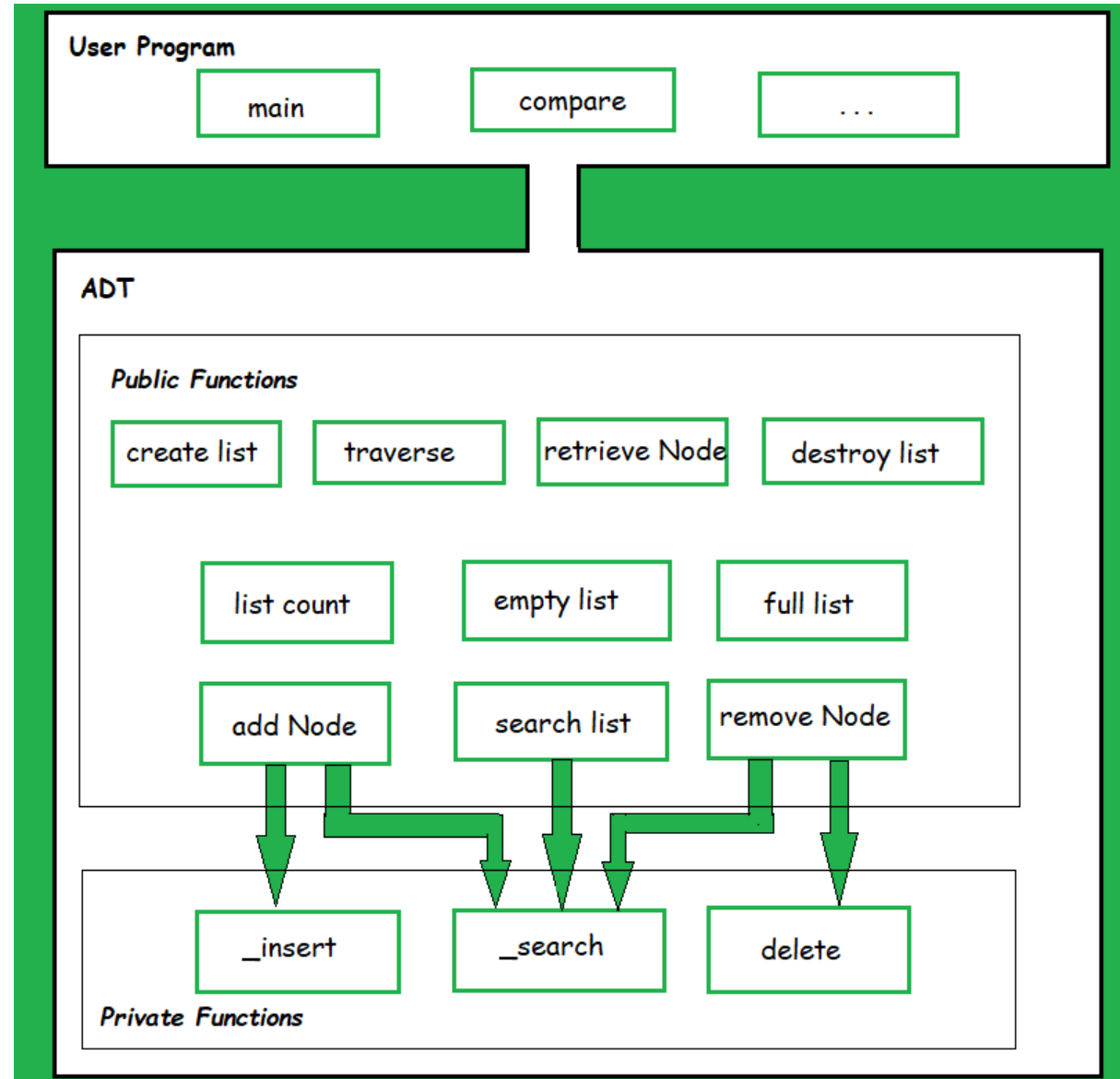
[Wikipedia]

O TAD LISTA

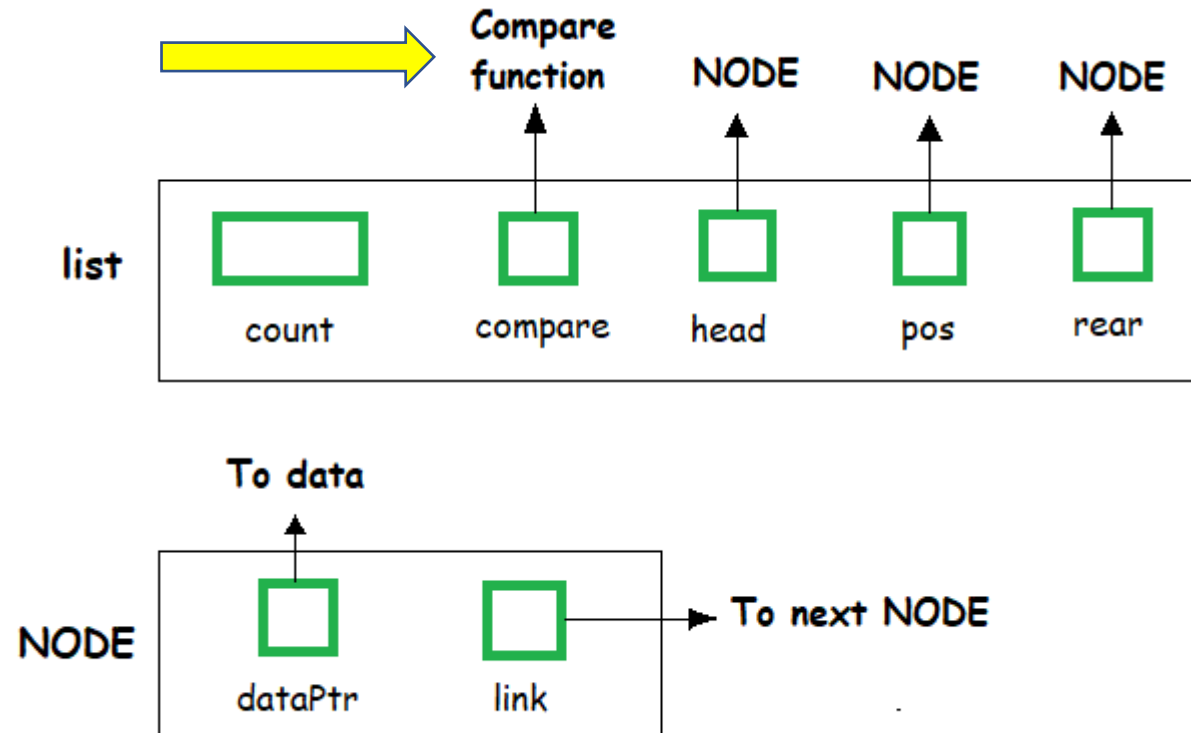
LISTA – Funcionalidades

- 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



Tarefa

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



[java2novice.com]

O TAD DEQUE

1ª 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:** considere as semelhanças com o TAD QUEUE implementado com um array circular

2ª 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