Tipos Abstratos de Dados II

22/10/2025

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

Sumário

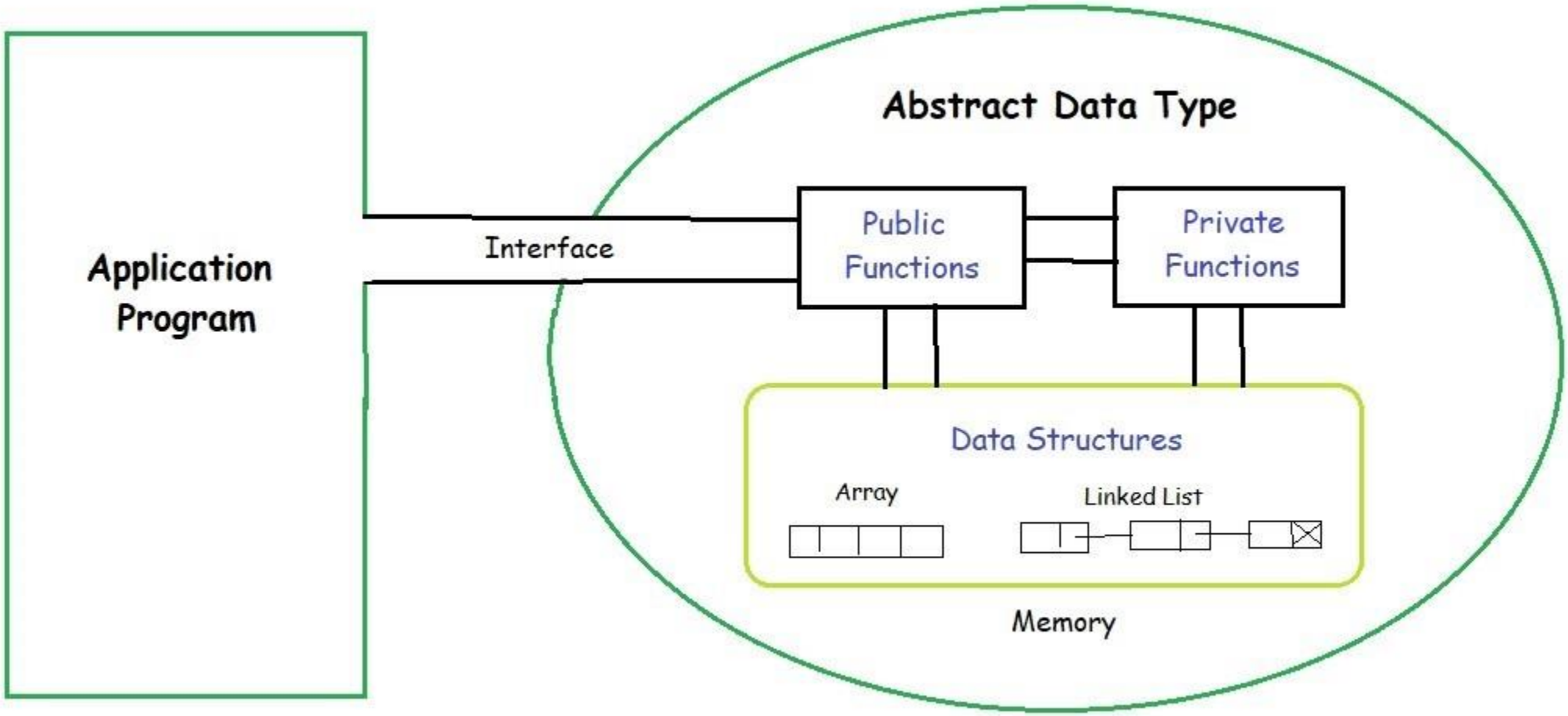
- Recap
- O TAD STACK de inteiros – Implementação usando um array
- O TAD STACK de ponteiros – Implementação usando um array
- O TAD QUEUE – Implementação usando um array circular
- O TAD DEQUE – Breve referência
- Exercícios / Tarefas 

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 um 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 o interface do TAD
  - Não é necessário alterar código que use o TAD !!

Resumo

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

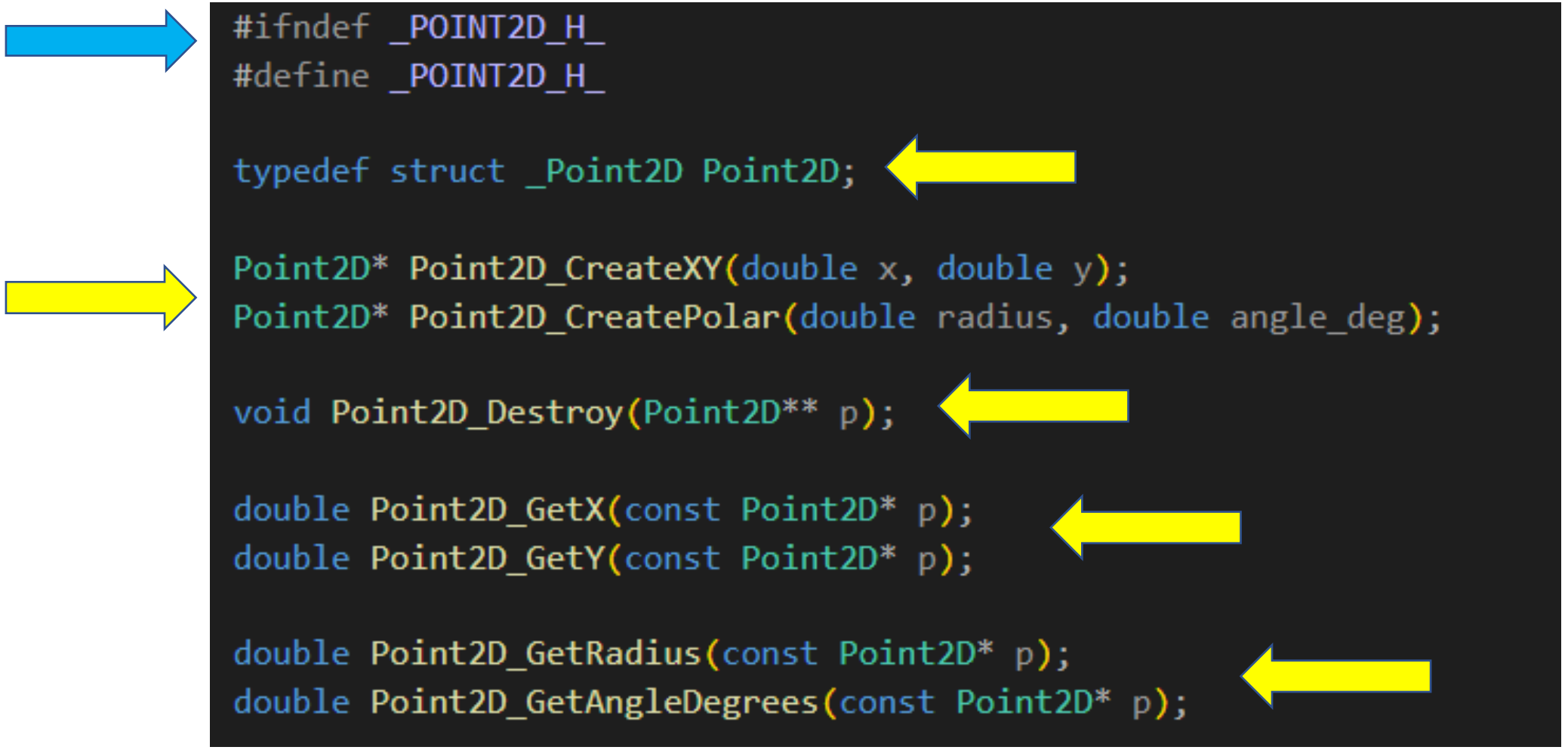
- Ficheiro **.h** : operações públicas + ponteiro para instância
- Ficheiro **.c** : implementação + representação interna

Tarefa – 2 versões do TAD Ponto 2D – Fizeram?

- Especificar uma só interface para o TAD Ponto 2D
- Desenvolver duas implementações distintas, com diferente representação interna (i.e., atributos)
  - Coordenadas cartesianas
  - Coordenadas polares
- A interface é a mesma para ambas as implementações
- Desenvolver um único programa de teste

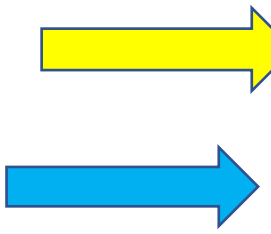
O TAD Ponto 2D

Point2D.h – Interface – 1 só ficheiro .h



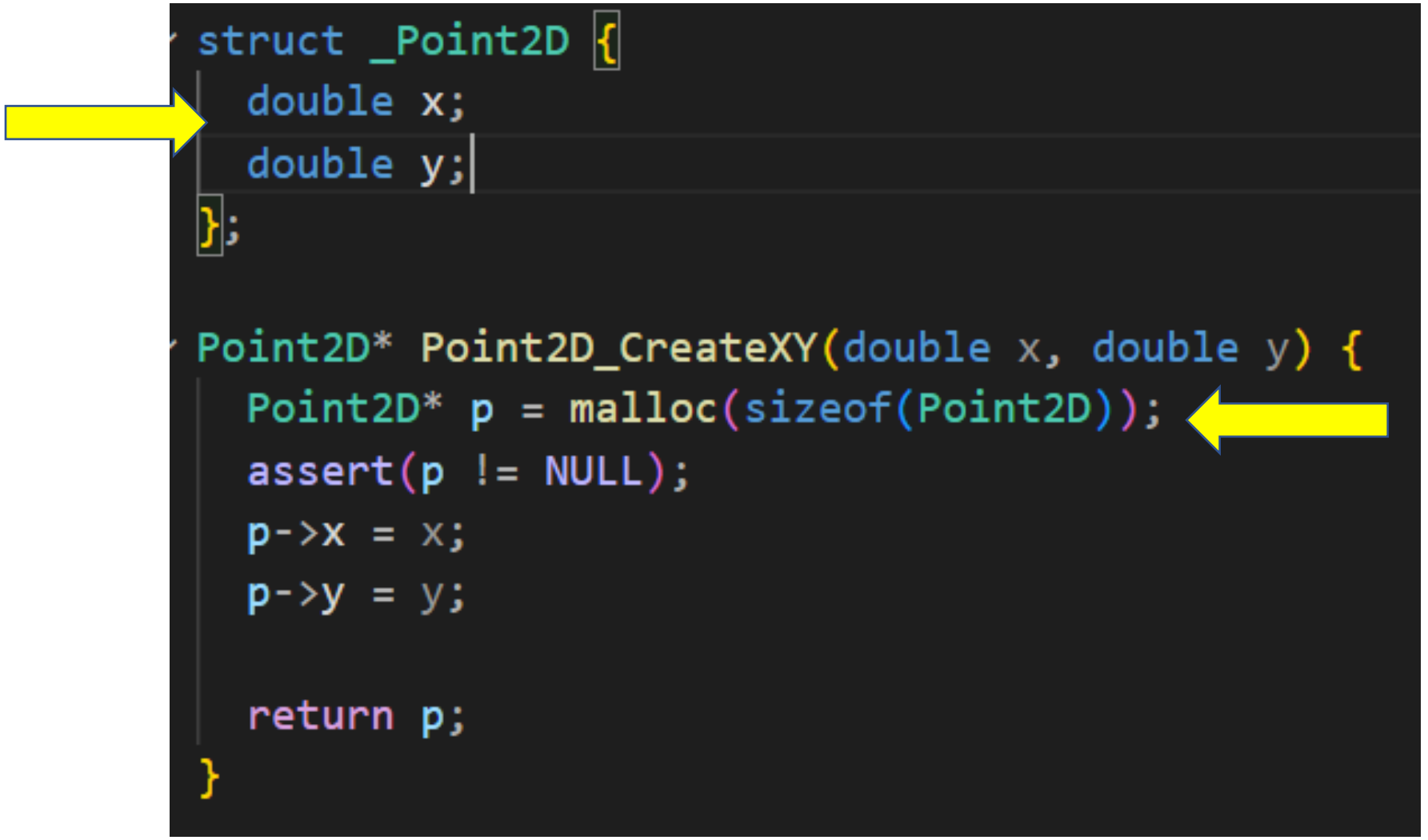
```
#ifndef _POINT2D_H_
#define _POINT2D_H_

typedef struct _Point2D Point2D;

Point2D* Point2D_CreateXY(double x, double y);
Point2D* Point2D_CreatePolar(double radius, double angle_deg);

void Point2D_Destroy(Point2D** p);

double Point2D_GetX(const Point2D* p);
double Point2D_GetY(const Point2D* p);

double Point2D_GetRadius(const Point2D* p);
double Point2D_GetAngleDegrees(const Point2D* p);
```

Point2D.h – Interface – 1 só ficheiro .h



```
int Point2D_IsEqual(const Point2D* p1, const Point2D* p2);
int Point2D_IsDifferent(const Point2D* p1, const Point2D* p2);

void Point2D_DisplayXY(const Point2D* p);
void Point2D_DisplayPolar(const Point2D* p);

double Point2D_Distance(const Point2D* p1, const Point2D* p2);

Point2D* Point2D_MidPoint(const Point2D* p1, const Point2D* p2);

#endif
```

Point2D_XY.c – Implementação – 1ª versão



```c
struct _Point2D {
    double x;
    double y;
};

Point2D* Point2D_CreateXY(double x, double y) {
    Point2D* p = malloc(sizeof(Point2D));
    assert(p != NULL);
    p->x = x;
    p->y = y;

    return p;
}
```

Point2D_Polar.c – Implementação – 2ª versão

```c
// radius >= 0
// angle in range [0, 2 * PI[
struct _Point2D {
  double radius;
  double angle;
};
```

```c
Point2D* Point2D_CreatePolar(double radius, double angle_deg) {
  assert(radius >= 0.0);
  assert((angle_deg >= 0.0) && (angle_deg < 360.0));

  Point2D* p = malloc(sizeof(Point2D));
  assert(p != NULL);
  p->radius = radius;
  p->angle = angle_deg * DEG_TO_RAD;

  return p;
}
```

Tarefas

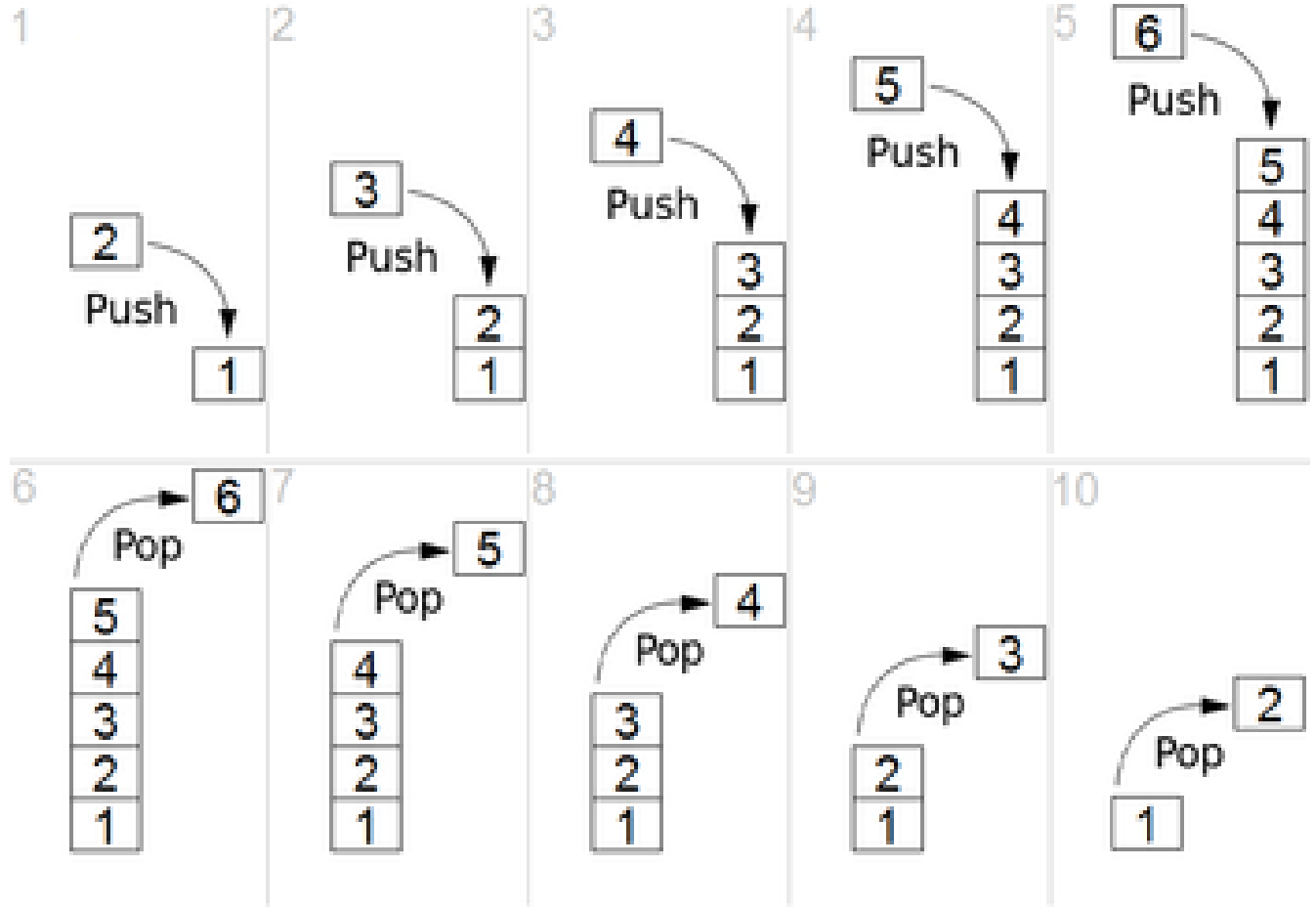
- Analisar os ficheiros disponibilizados !!
- 1 só ficheiro de teste : testing_Point2D.c
- Compilação

```
gcc -Wall -Wextra testing_Point2D.c Point2D_XY.c -o testing_Point2D_XY -lm

gcc -Wall -Wextra testing_Point2D.c Point2D_Polar.c -o testing_Point2D_Polar -lm
```

O TAD STACK / PILHA

[Wikipedia]

STACK / PILHA – Push & Pop



[Wikipedia]

STACK / PILHA – Funcionalidades

- Conjunto de elementos do mesmo tipo
- Armazenados em ordem sequencial
- LIFO : Last-In-First-Out
  - 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

```c
#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

```c
#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 – Construtor & Destrutor

```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;
}
```

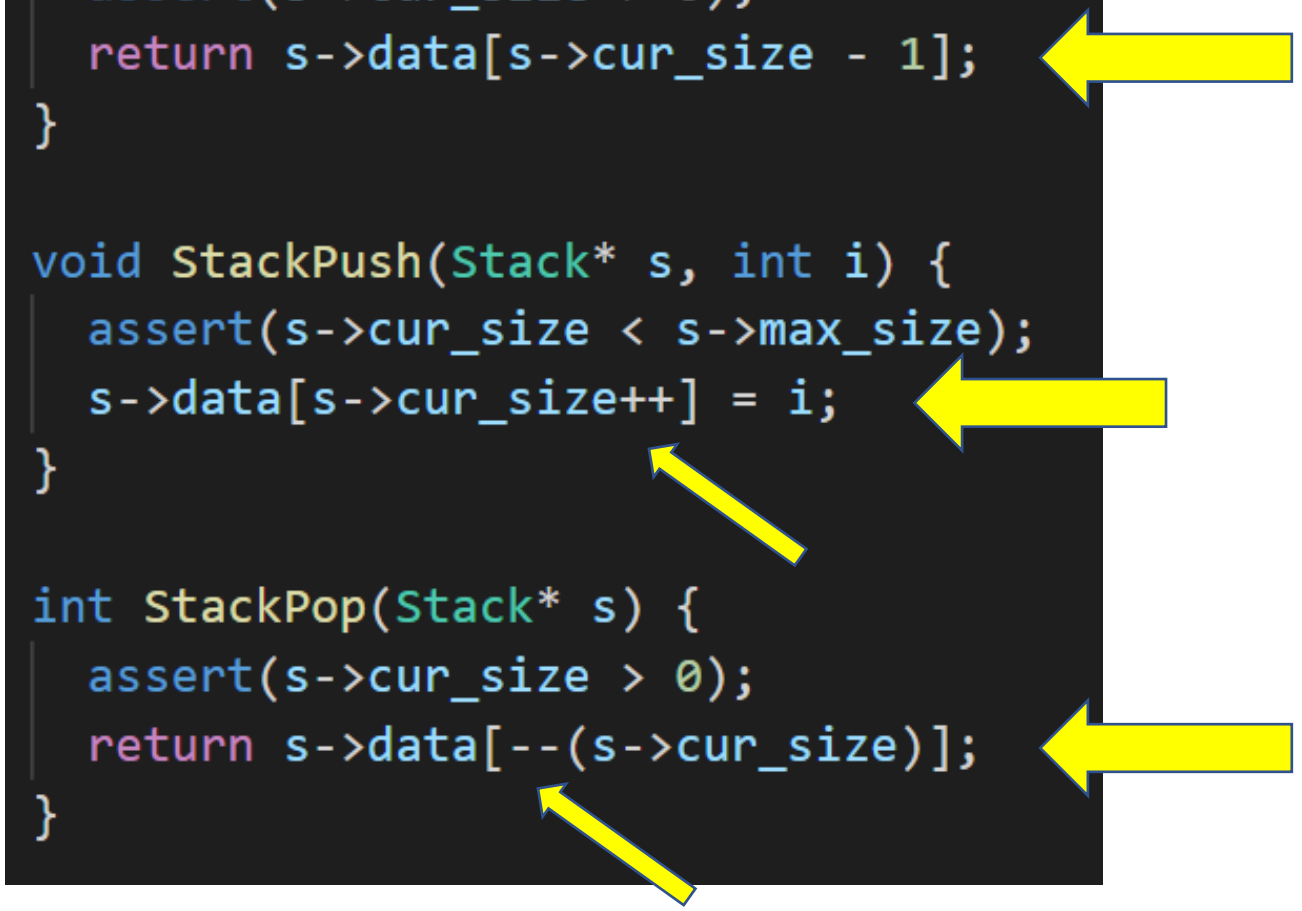
```c
void StackDestroy(Stack** p) {
  assert(*p != NULL);
  Stack* s = *p;
  free(s->data);
  free(s);
  *p = NULL;
}
```

IntegersStack.c – Operações mais simples

```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 – Peek – Push – Pop

```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)];
}
```



Tarefa – Aplicação

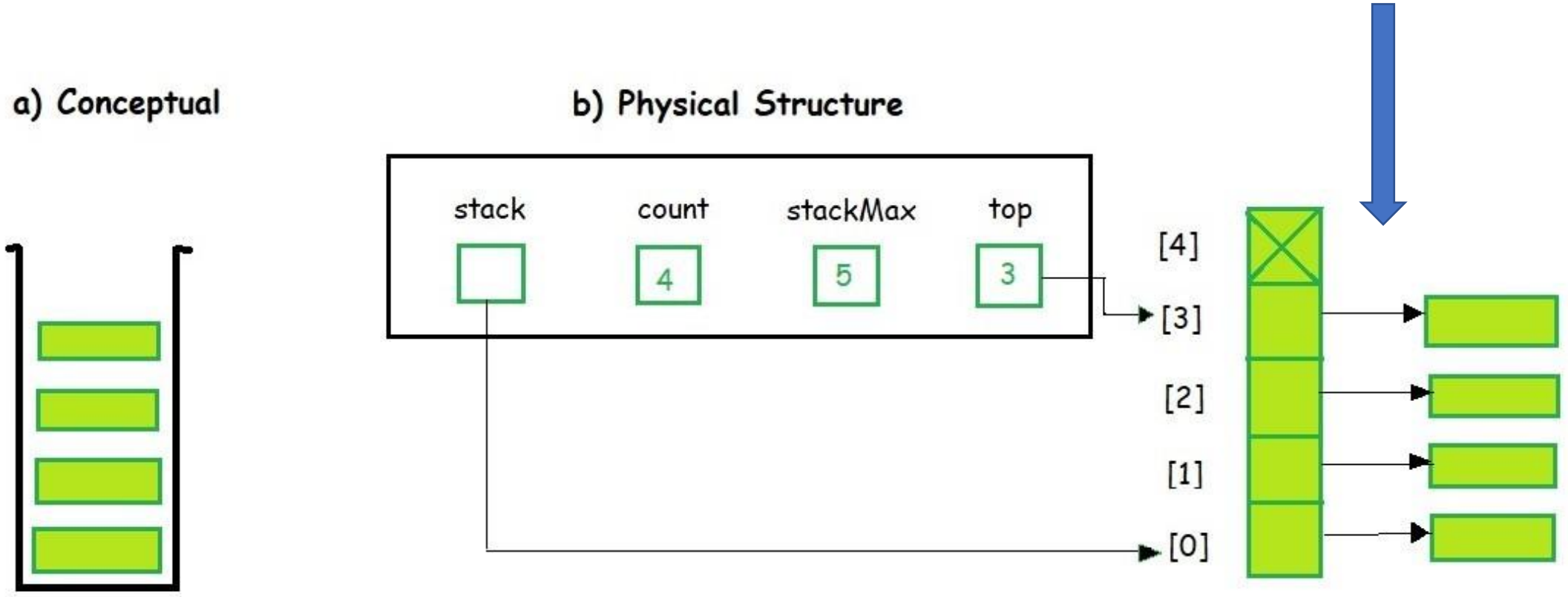
- Como escrever pela ordem inversa os algarismos de um número inteiro positivo ?
- Como se pode utilizar o TAD STACK ?
- **TAREFA : Analisar o exemplo de aplicação !!**

O TAD STACK / PILHA

[Wikipedia]

- Array de Ponteiros Genéricos

O TAD Stack – Array de ponteiros



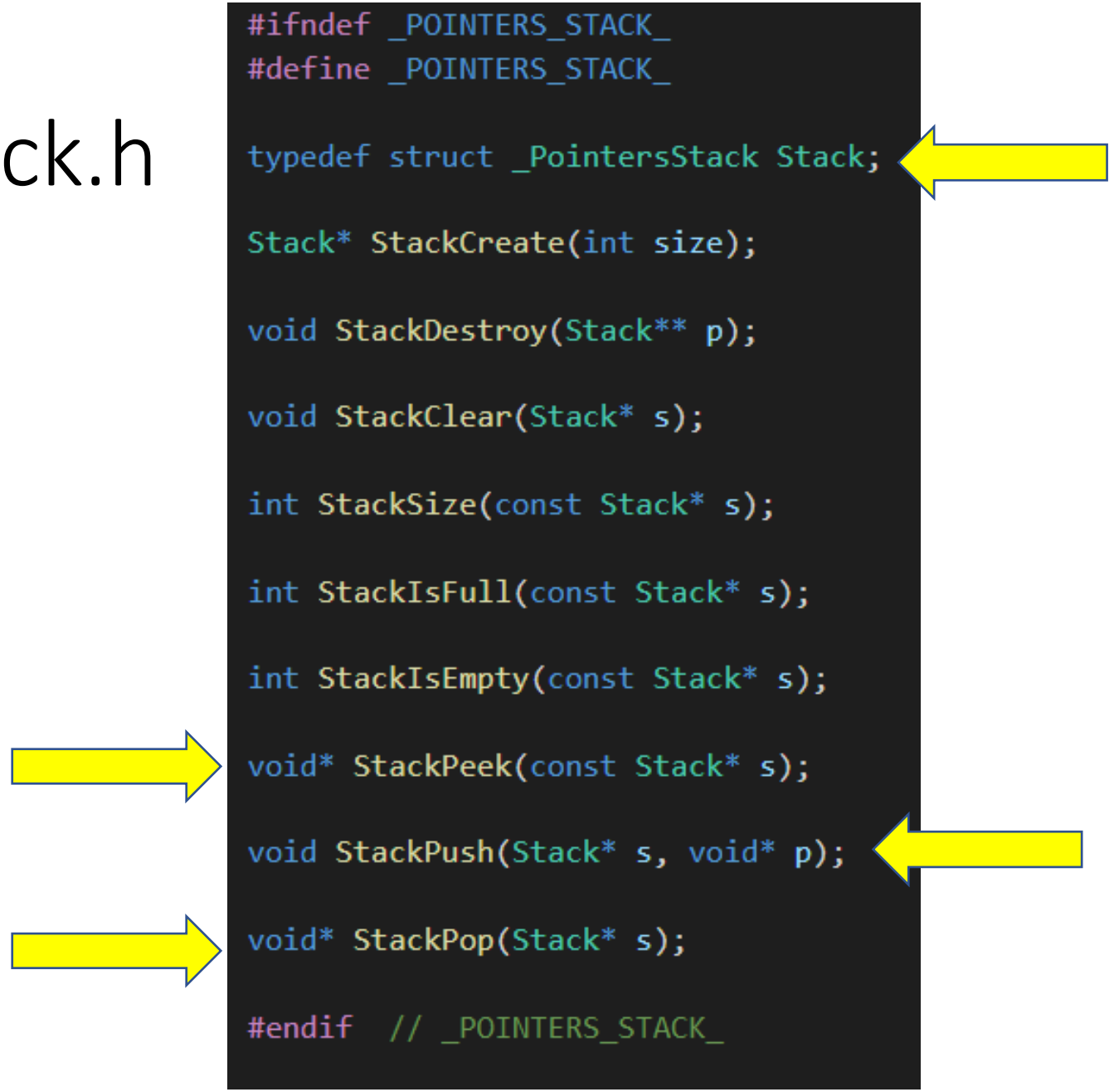
PointersStack.h

- Alterações ?

```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.h

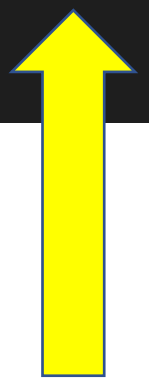
- Alterações ?



```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 – Array de ponteiros

```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 ?

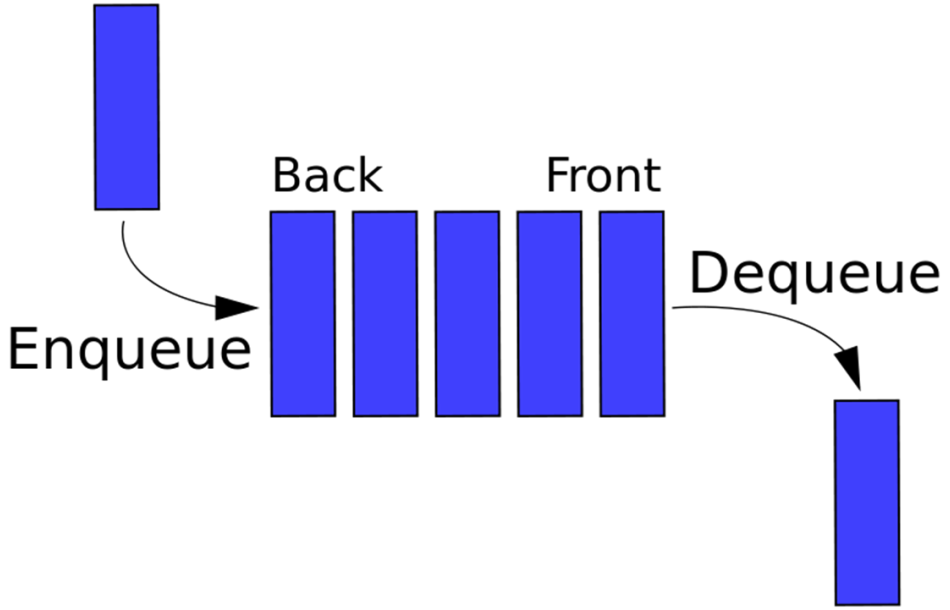
Tarefa – Aplicação

- 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 !!**

Stack de pontos 2D

- Como se pode utilizar esta nova versão do TAD STACK ?

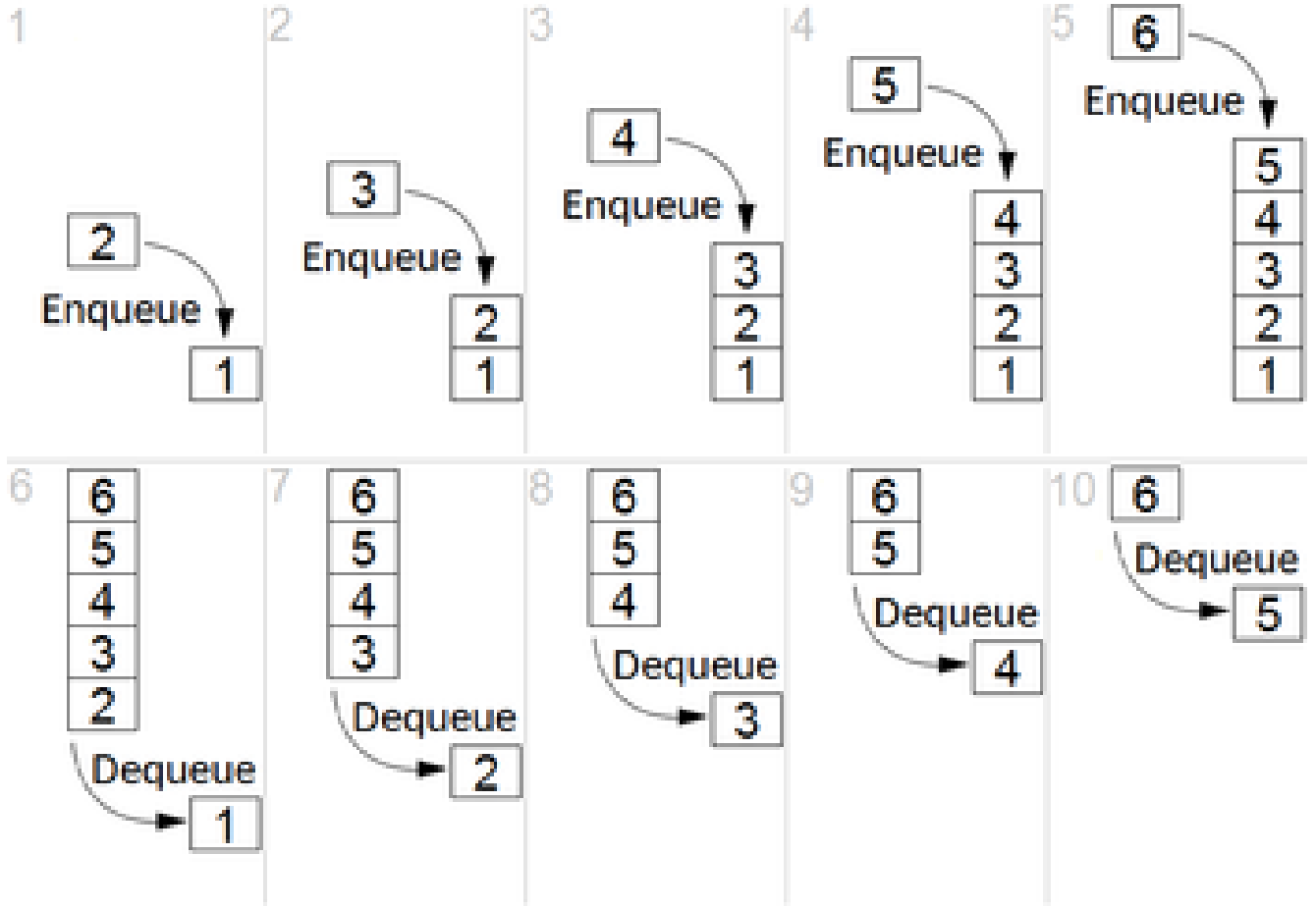
- **TAREFA : Analisar o exemplo de aplicação !!**



[Wikipedia]

O TAD QUEUE / FILA

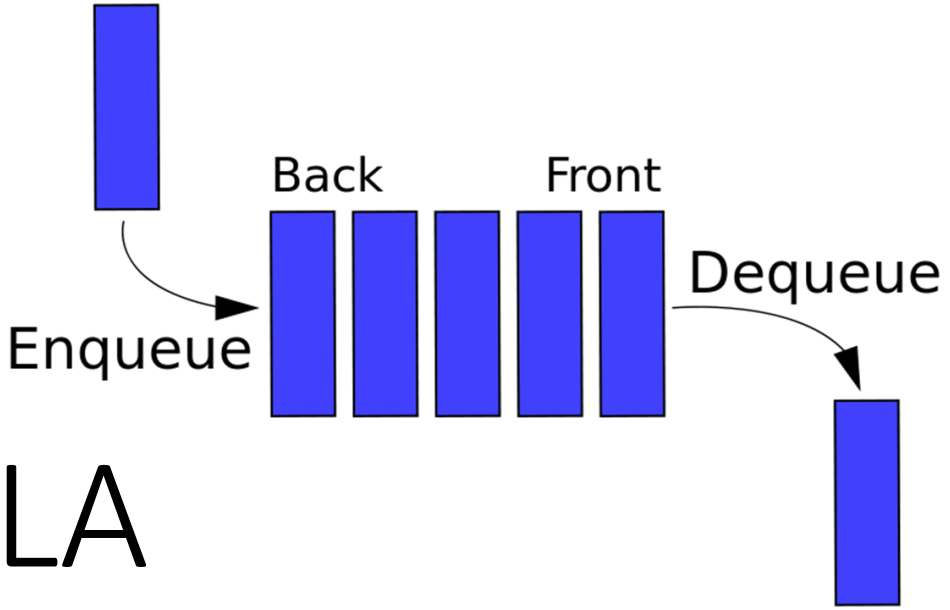
QUEUE / FILA – Enqueue & Dequeue



[Wikipedia]

QUEUE / FILA – Funcionalidades

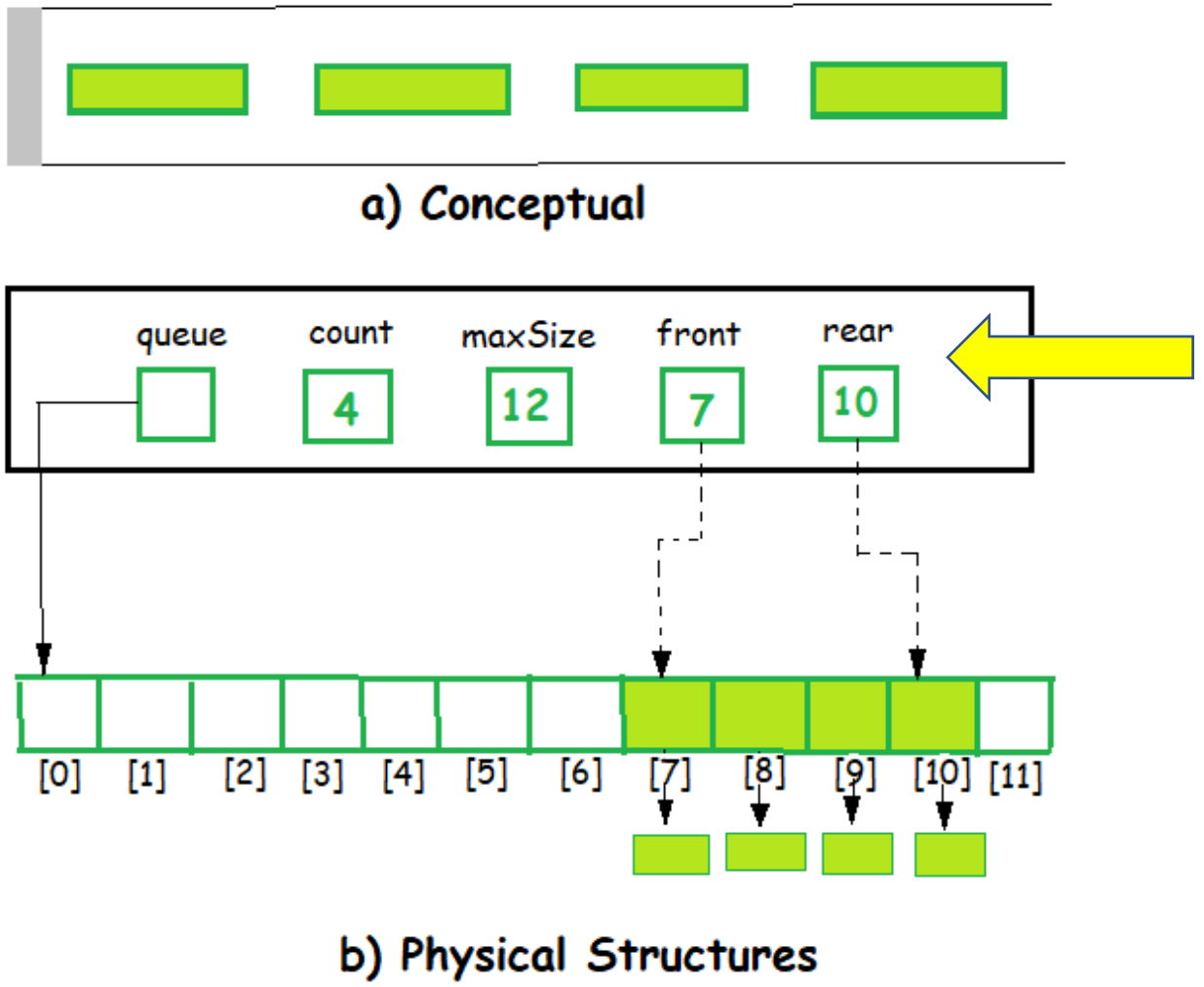
- Conjunto de elementos do mesmo tipo
- Armazenados em ordem sequencial
- FIFO : First-In-First-Out
  - Inserção na cauda da fila
  - Remoção / consulta apenas na frente da fila
- enqueue() / dequeue() / peek()
- size() / isEmpty() / isFull()
- init() / destroy() / clear()



[Wikipedia]

O TAD QUEUE / FILA
- Array Circular de Ponteiros

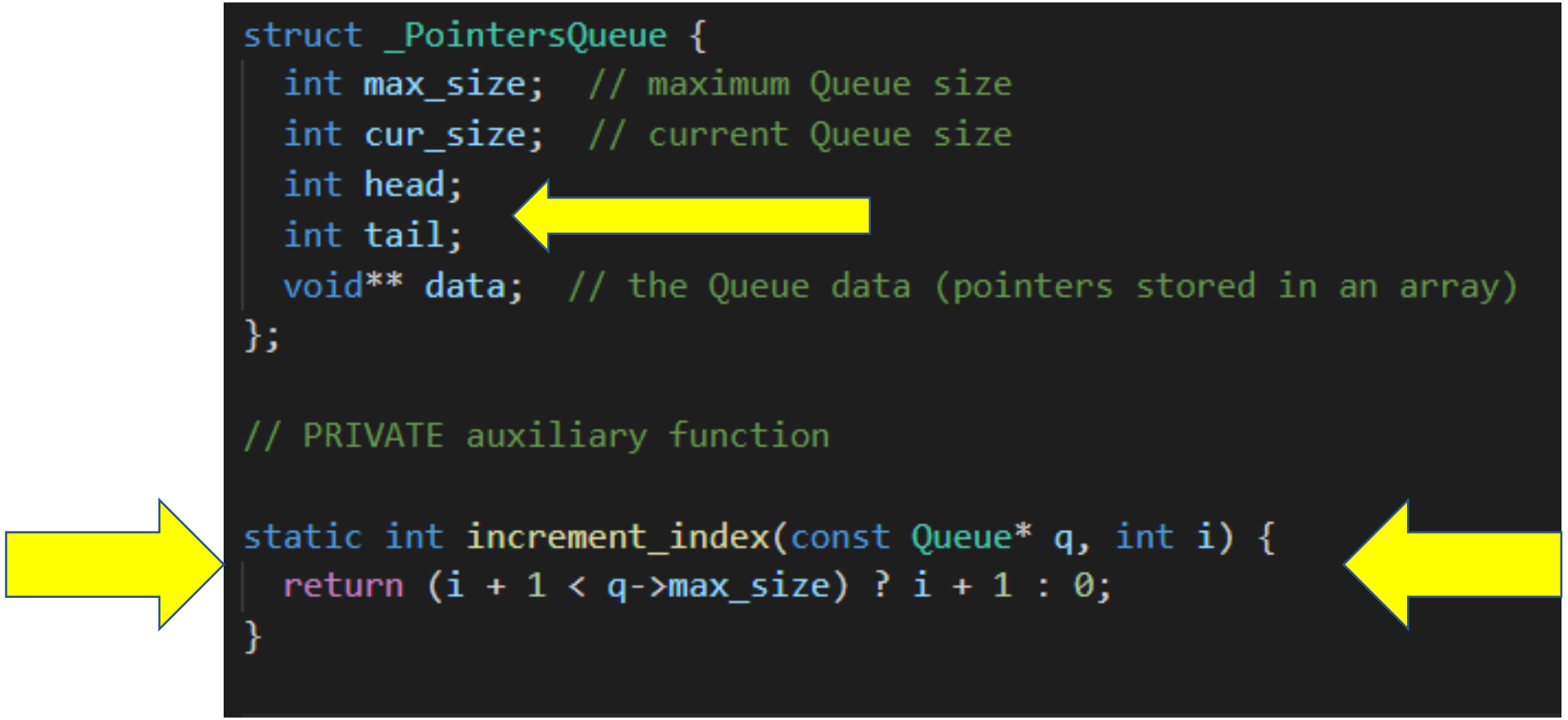
O TAD QUEUE – Array circular de ponteiros



PointersQueue.h

```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_
```

PointersQueue.c – Indexar um array circular



```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 – Construtor & Destrutor

```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;
}
```

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

PointersQueue.c – Enqueue – Dequeue

```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++;
}
```

```c
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];
}
```

Tarefa – PointersQueue.c

- **TAREFA : Analisar a implementação das funções do TAD**
- Analisar a implementação das operações sobre o array circular
- **TAREFA : Analisar o exemplo de aplicação !**

[java2novice.com]

O TAD DEQUE
- Double-Ended Queue

DEQUE – Funcionalidades

- Conjunto de elementos do mesmo tipo
- Armazenados em ordem sequencial

- Inserção / remoção / consulta nas duas extremidades

- add_at_front() / remove_at_front() / peek_at_front()
- add_at_rear () / remove_at_rear() / peek_at_rear()
- size() / isEmpty() / isFull()
- init() / destroy() / clear()

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