DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Projecto 2

IAED, 2014/2015

Processamento de cheques

- O programa irá permitir o registo e processamento de cheques emitidos entre clientes de um banco.
- Cada cheque tem um emissor, um beneficiário e um valor.

Valor: _____ **$**
(montante a transferir em `int`)

Emissor: _____
(referência do cliente em `long int`)

Beneficiário: _____
(referência do cliente em `long int`)

Cheque: _____
(referência do cheque em `long int`)

Funcionalidades do programa

- Registo de cheques de determinado valor, emitidos por um dado cliente, com dado beneficiário.
  Comando: `cheque valor refe refb refc`
- Processamento do cheque mais antigo, ou aquele com uma dada referência
  Comandos: `processa` e `processaR refc`
- Obtenção de informação sobre um dado cheque, cliente, ou clientes activos
  Comandos: `infocheque refc`, `infocliente refc` e `info`
- Fecho do programa acompanhado da eliminação de todo o registo de dados.
  Comando: `sair`

1º - PENSAR

Como organizar os dados?

- Colecção de cheques?

Essa organização deve permitir

  - Introduzir um novo cheque na colecção
  - Eliminar um cheque da colecção
  - Obter o cheque emitido há mais tempo (fila de espera)
  - Obter um cheque com uma dada referência (tabela de símbolos)

- Colecção de clientes?

Essa organização deve permitir

  - Obter informação sobre todos os cheques que envolvam um dado cliente
  - Determinar o conjunto (ordenado) de clientes activos

Solução mais simples: basear tudo numa colecção de cheques.

Será que conseguimos fazer melhor?

Como estruturar os dados?

– Não existe limite para o número de cheques ou clientes.
  ➡ Usar estruturas de dados dinâmicas!
    - Listas (simplesmente ligadas? indexadas ao início e fim?...)
    - Árvores binárias (de pesquisa? equilibradas?)
    - Tabelas de dispersão?
    - Outra?
– Não se conhece a frequência de utilização de cada comando.
  ➡ Procurar soluções que permitam lidar com todos os comandos eficientemente.

Eficiência - para o 16!

- A eficiência vai ser avaliada (componente avaliação automática 8/16 testes).
- A escolha das estruturas de dados vai ter impacto na eficiência que se pode atingir.

Entidade única: cheque (usando listas, árvores, AVLs, tabelas de hashing...)

- Complexidade da inserção de cheques?
- Complexidade da pesquisa de cheques por ordem de inserção? e por chave?
- Complexidade da recolha de informação sobre clientes?

Duas entidades: cheque e cliente

- Complexidade da inserção de cheques?
- Complexidade da pesquisa de cheques por tempo? e por chave?
- Complexidade da recolha de informação sobre clientes?

Abstracção de dados - para o 20! ☺

- O uso de ADTs vai ser avaliado (componente qualidade de código).

Separação entre a implementação do ADT e o cliente

- Definição do interface no `ADT.h`, implementação no `ADT.c`
- Cliente apenas usa funções definidas no `ADT.h`

➡ Verificação: posso substituir a implementação `ADT1.c` pela `ADT2.c` (ambas compatíveis com `ADT.h`), e o programa continua a funcionar?

Separação entre o tipo de objectos quardados e a definição e implementação do ADT

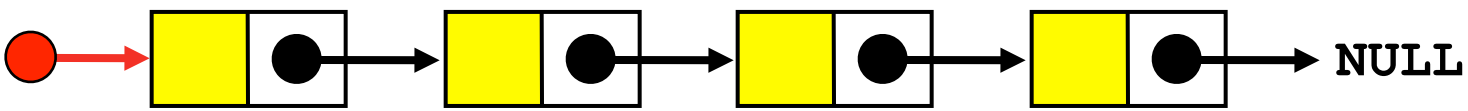
- Definição do interface no `Item.h`, implementação no `Item.c`

➡ Verificação: posso substituir o par `Item1.h`/`Item1.c` pelo par `Item2.h`/ `Item2.c` (ambos compatíveis com `ADT.h`), e o programa continua a funcionar?

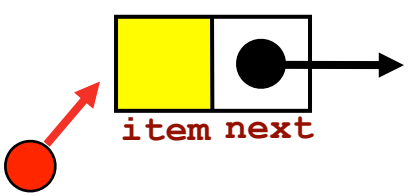
2º PROGRAMAR

ADT: Estruturas possíveis (ver aulas!)

Listas de Itens...




```
typedef struct node{
   Item item;
   struct node *next;
}* link;
```

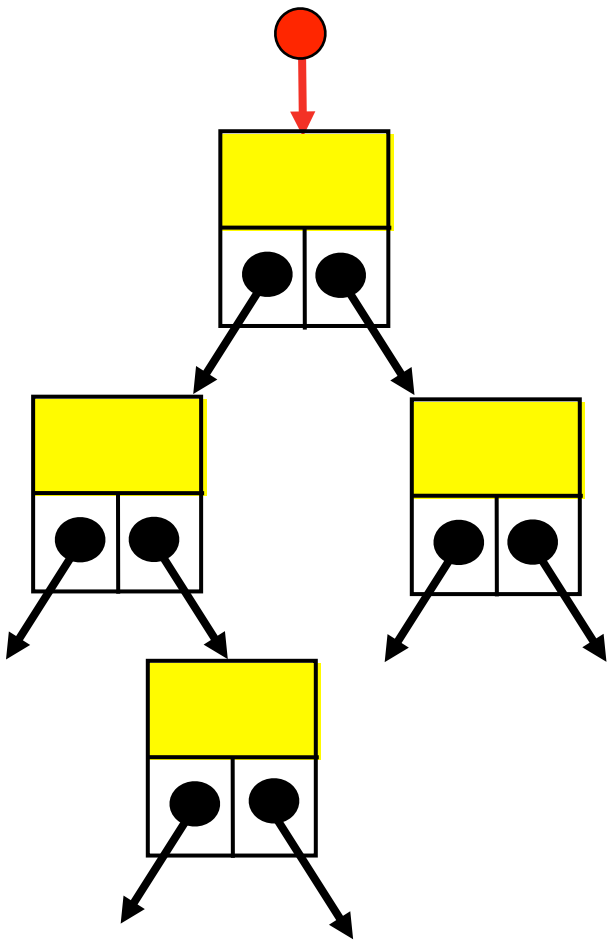


Também pode ser útil:

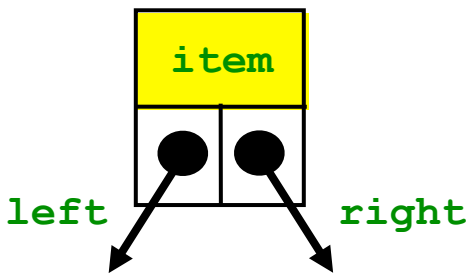
```
typedef struct queue{
   link head, tail;
}* Q;
```

ADT: Estruturas possíveis (ver aulas!)

Árvores (equilibradas?) de Items... ETC!

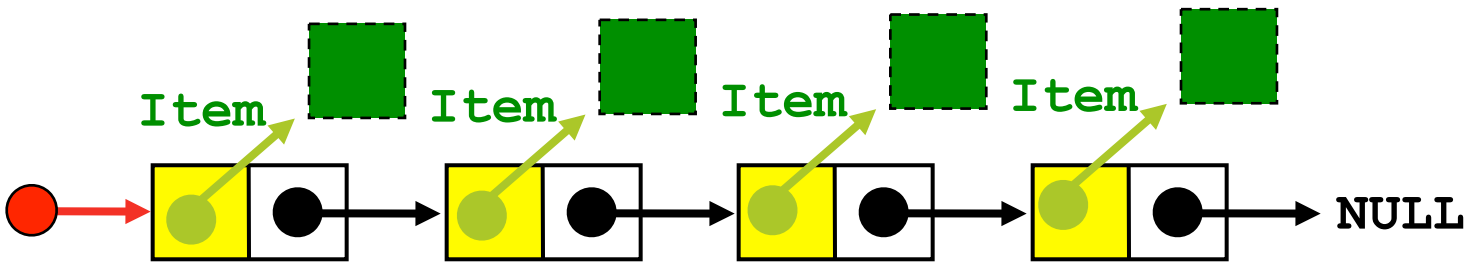


```
typedef struct node{
   Item item;
   struct node *left, *right;
}* link;
```

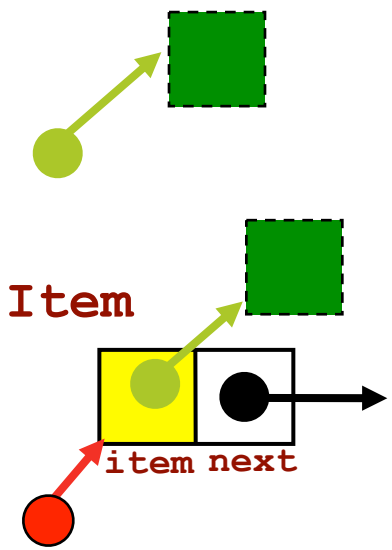


ADT: Estruturas possíveis (ver aulas!)

Podemos definir a estrutura de dados de forma independente do que é um Item

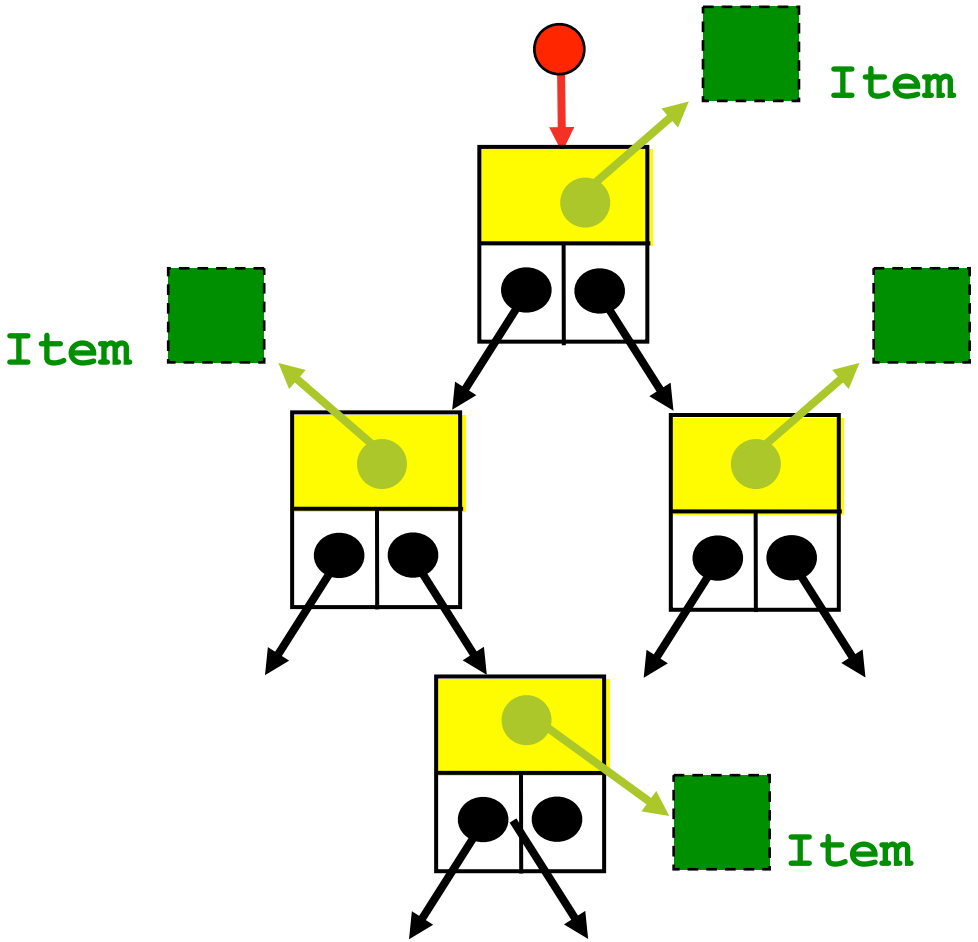



```
typedef struct {
   (referencia, valor,
    cheque...)
}* Item;


typedef struct node{
    Item item;
    struct node *next;
}* link;
```

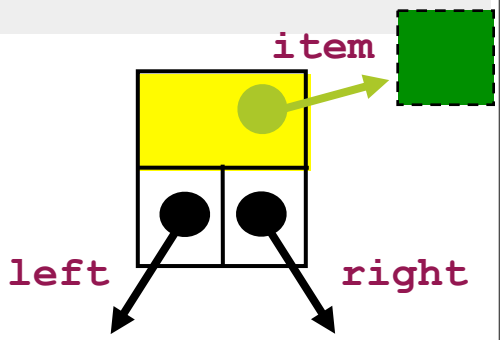


ADT: Estruturas possíveis (ver aulas!)

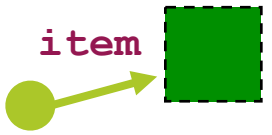
Podemos definir a estrutura de dados de forma independente do que é um Item




```
typedef struct node{
   Item item;
   struct node *left, *right;
}* link;
```




```
typedef struct mensagem{
   (referencia, valor,
    cheque...)
}*Item;
```



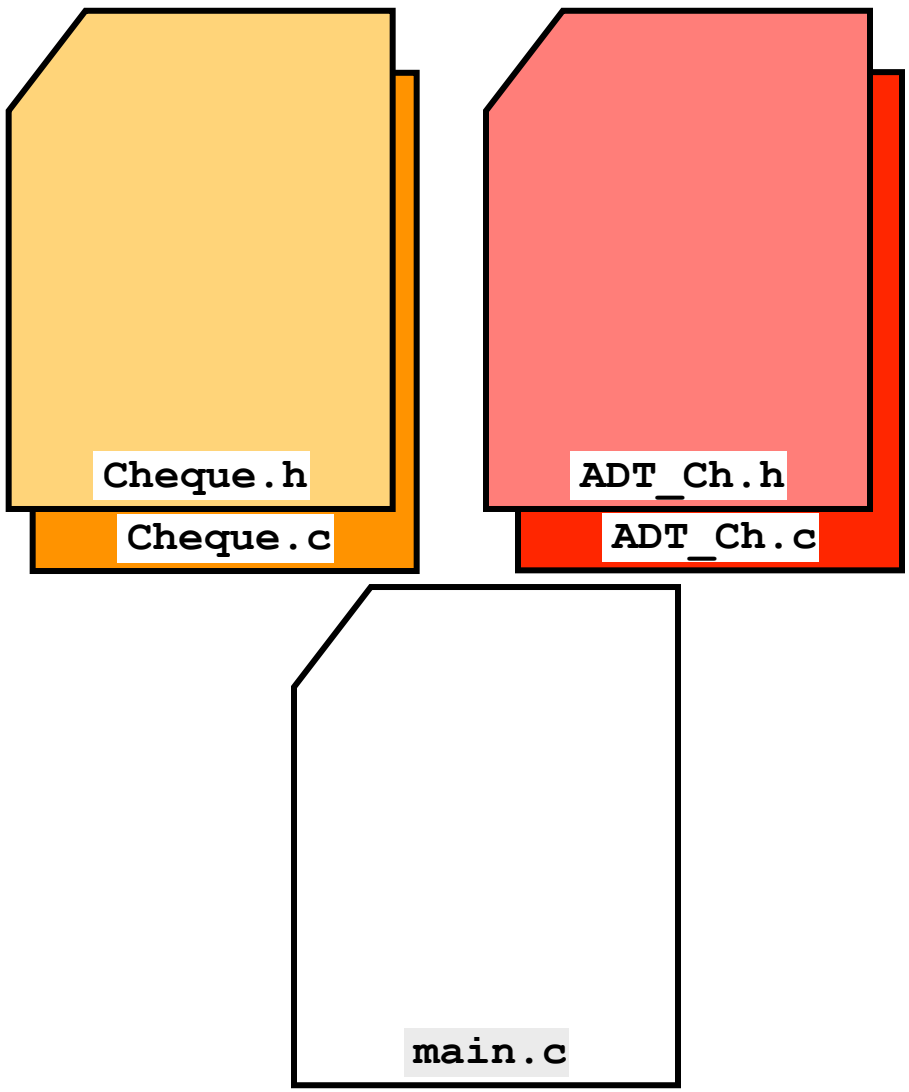
Qualidade do código e outras dicas

- Idente e documente convenientemente o seu código.
- Organize o código da melhor forma, se possível em vários ficheiros.
- Elimine eventuais fugas de memória (valgrind).

- Teste o seu código à medida que o escreve.
- Submeta com antecedência.

Bons códigos!

Sugestão de estrutura (I)

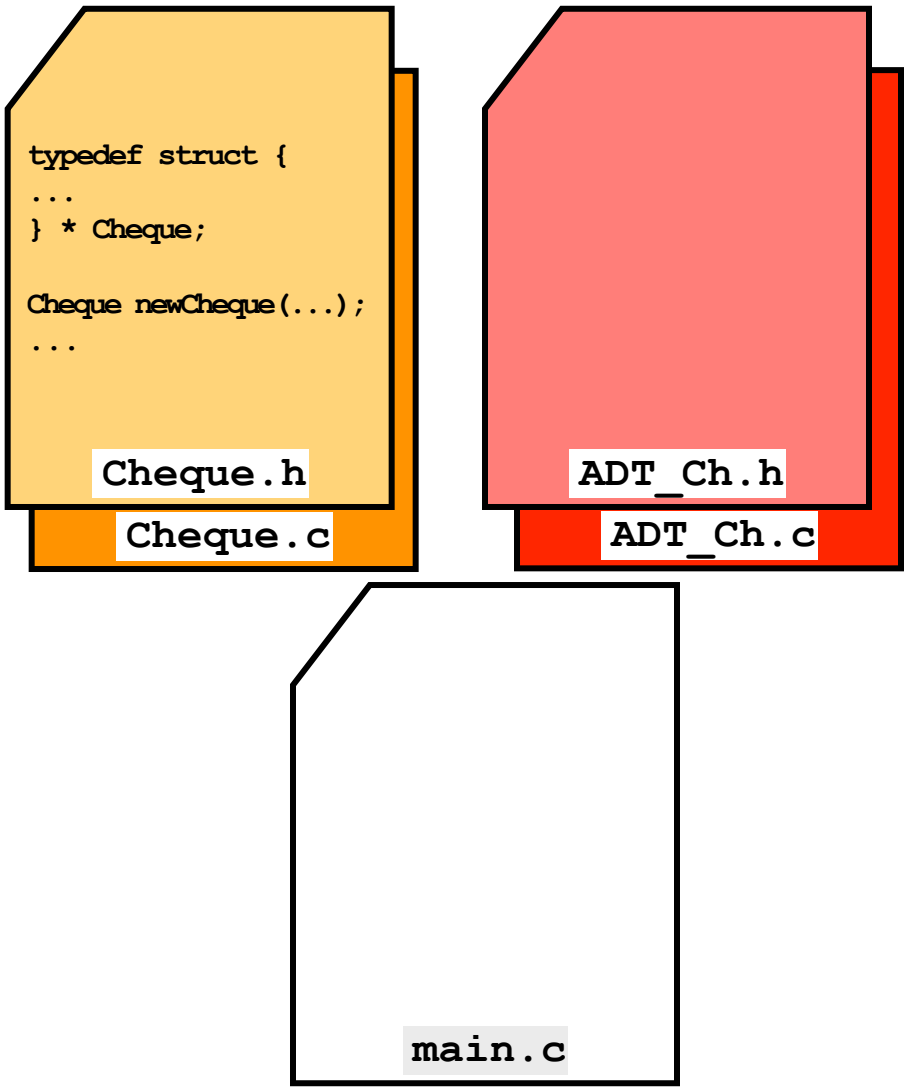


Entidade única - cheque:

- `Cheque.h` declara as funcionalidades do "Item" `Cheque`
- `Cheque.c` implementa as funcionalidades do "Item" `Cheque`
- `ADT_Ch.h` declara as funcionalidades do ADT que organiza `Cheque`
- `ADT_Ch.c` implementa as funcionalidades do ADT que organiza `Cheque`
- `main.c` contém o programa que resolve o problema utilizando as funcionalidades disponibilizadas em `Cheque.h` e `ADT_Ch.h`

```
$ gcc -ansi -pedantic -Wall main.c Cheque.c
```

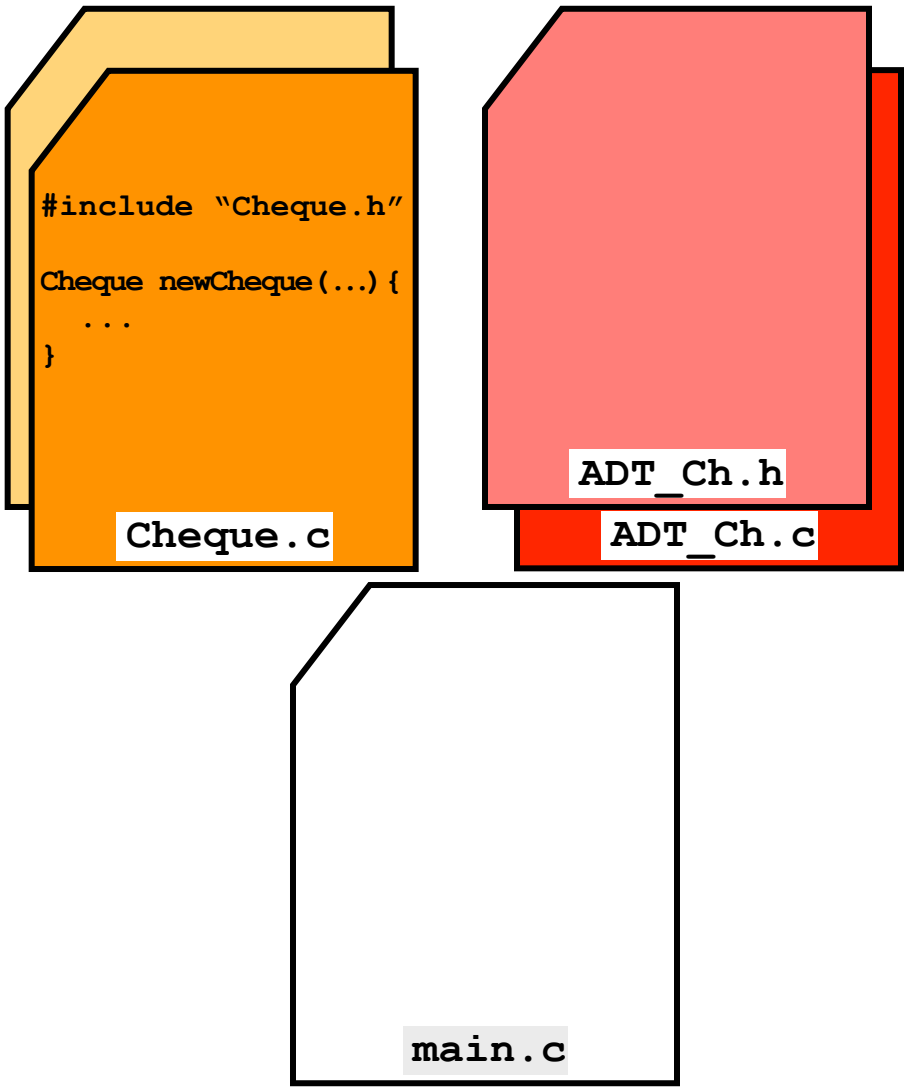
Sugestão de estrutura (I)



Entidade única - cheque:

- `Cheque.h` declara as funcionalidades do "Item" `Cheque`
- `Cheque.c` implementa as funcionalidades do "Item" `Cheque`
- `ADT_Ch.h` declara as funcionalidades do ADT que organiza `Cheque`
- `ADT_Ch.c` implementa as funcionalidades do ADT que organiza `Cheque`
- `main.c` contém o programa que resolve o problema utilizando as funcionalidades disponibilizadas em `Cheque.h` e `ADT_Ch.h`

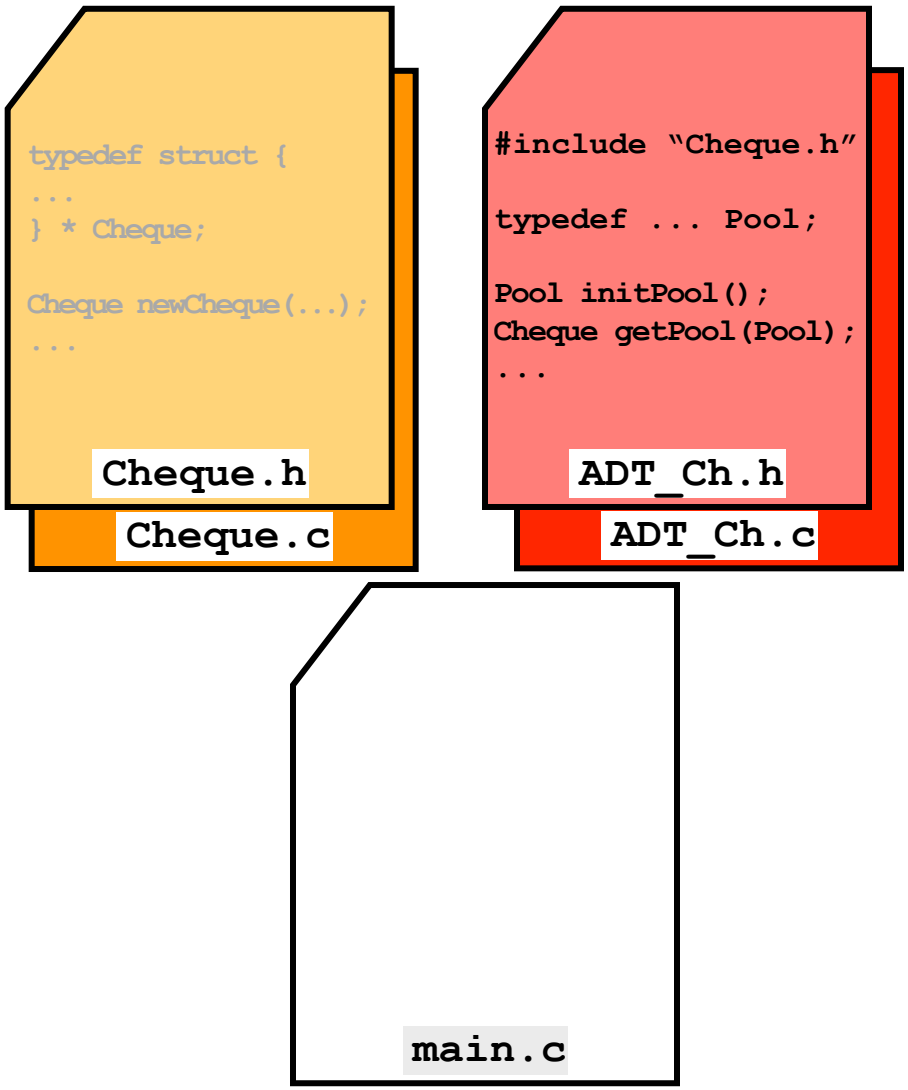
Sugestão de estrutura (I)



Entidade única - cheque:

- `Cheque.h` declara as funcionalidades do "Item" `Cheque`
- `Cheque.c` implementa as funcionalidades do "Item" `Cheque`
- `ADT_Ch.h` declara as funcionalidades do ADT que organiza `Cheque`
- `ADT_Ch.c` implementa as funcionalidades do ADT que organiza `Cheque`
- `main.c` contém o programa que resolve o problema utilizando as funcionalidades disponibilizadas em `Cheque.h` e `ADT_Ch.h`

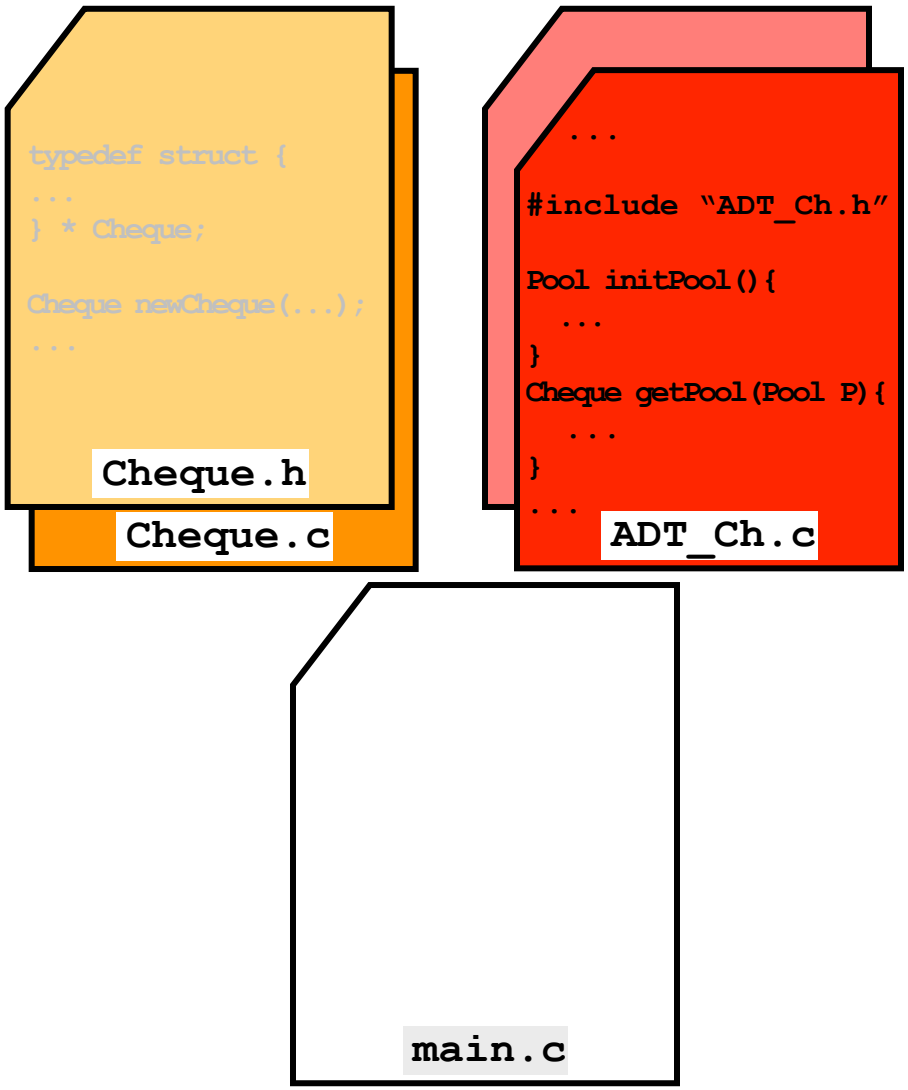
Sugestão de estrutura (I)



Entidade única - cheque:

- `Cheque.h` declara as funcionalidades do "Item" `Cheque`
- `Cheque.c` implementa as funcionalidades do "Item" `Cheque`
- `ADT_Ch.h` declara as funcionalidades do ADT que organiza `Cheque`
- `ADT_Ch.c` implementa as funcionalidades do ADT que organiza `Cheque`
- `main.c` contém o programa que resolve o problema utilizando as funcionalidades disponibilizadas em `Cheque.h` e `ADT_Ch.h`

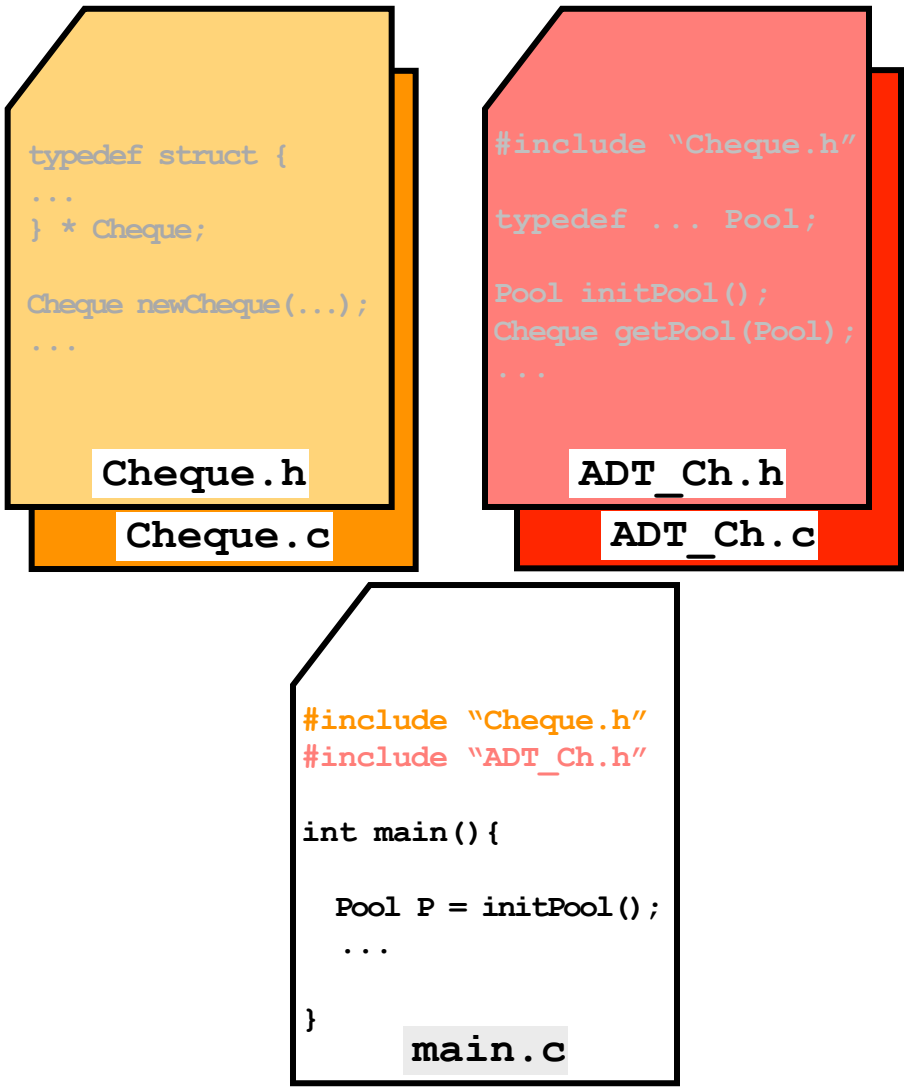
Sugestão de estrutura (I)



Entidade única - cheque:

- `Cheque.h` declara as funcionalidades do "Item" `Cheque`
- `Cheque.c` implementa as funcionalidades do "Item" `Cheque`
- `ADT_Ch.h` declara as funcionalidades do ADT que organiza `Cheque`
- `ADT_Ch.c` implementa as funcionalidades do ADT que organiza `Cheque`
- `main.c` contém o programa que resolve o problema utilizando as funcionalidades disponibilizadas em `Cheque.h` e `ADT_Ch.h`

Sugestão de estrutura (I)



Entidade única - cheque:

- `Cheque.h` declara as funcionalidades do "Item" `Cheque`
- `Cheque.c` implementa as funcionalidades do "Item" `Cheque`
- `ADT_Ch.h` declara as funcionalidades do ADT que organiza `Cheque`
- `ADT_Ch.c` implementa as funcionalidades do ADT que organiza `Cheque`
- `main.c` contém o programa que resolve o problema utilizando as funcionalidades disponibilizadas em `Cheque.h` e `ADT_Ch.h`

Sugestão de estrutura (I)

```
typedef struct {
...
} * Cheque;

Cheque newCheque(...);
...
```

Cheque.h

Cheque.c

```
#include "Cheque.h"

typedef ... Pool;

Pool initPool();
Cheque getPool(Pool);
...
```

ADT_Ch.h

ADT_Ch.c

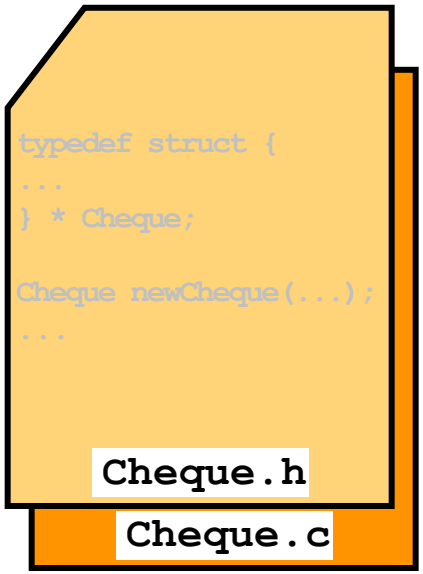
```
#include "Cheque.h"
#include "ADT_Ch.h"

int main(){

  Pool P = initPool();
   ...

}
```

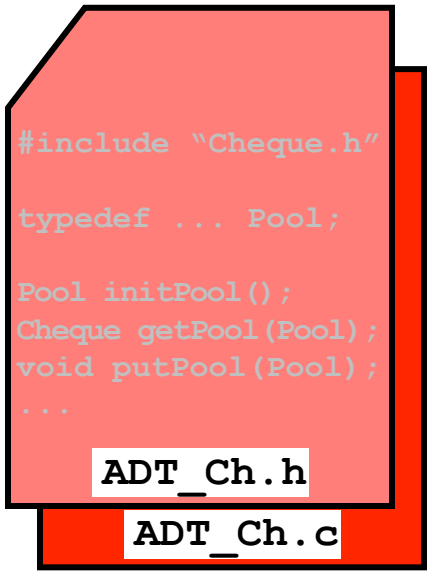
main.c

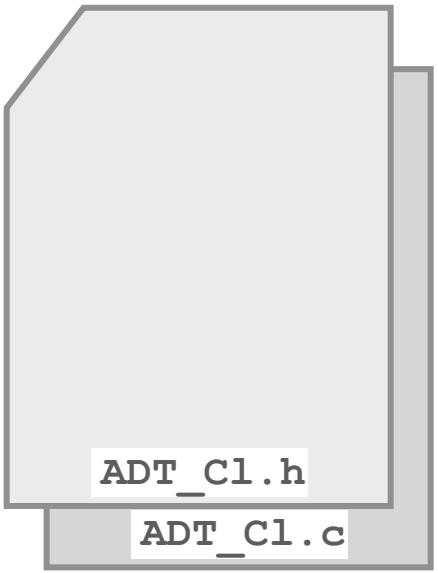
Compilar todos os .c

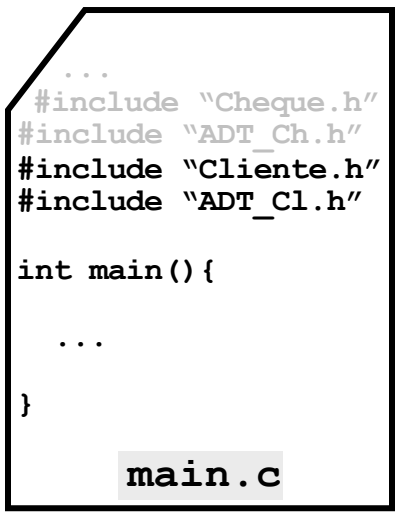
```
$ gcc -ansi -pedantic -Wall main.c Cheque.c ADT_Ch.c
```

Sugestão de estrutura (II)





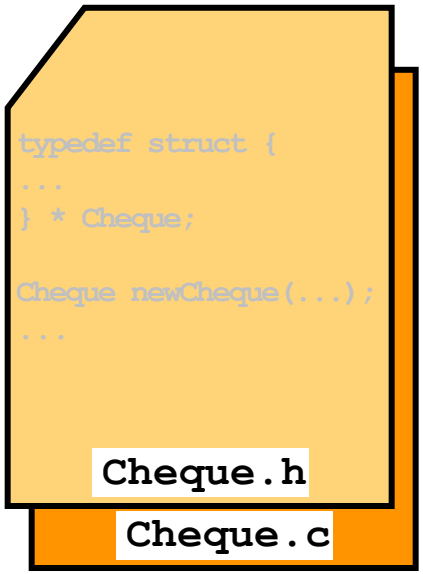


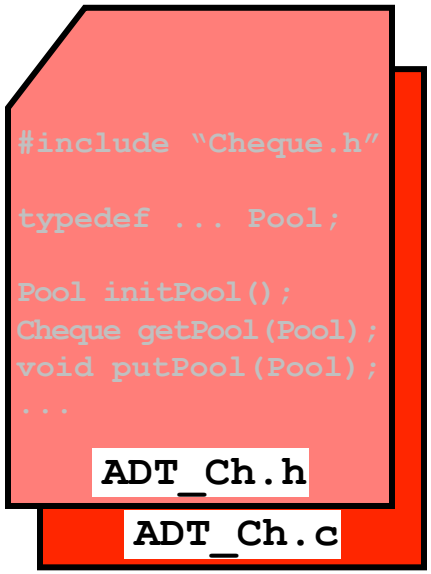


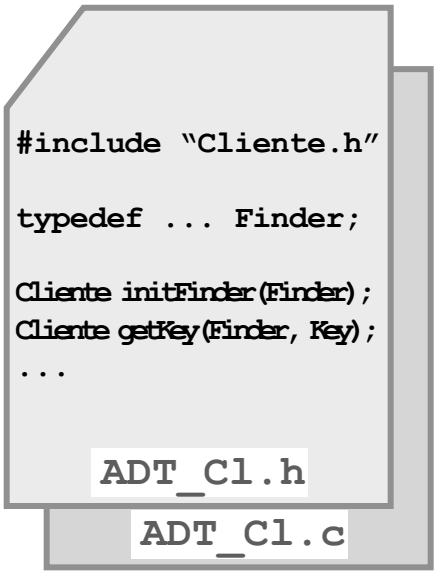
Entidade adicional - cliente:

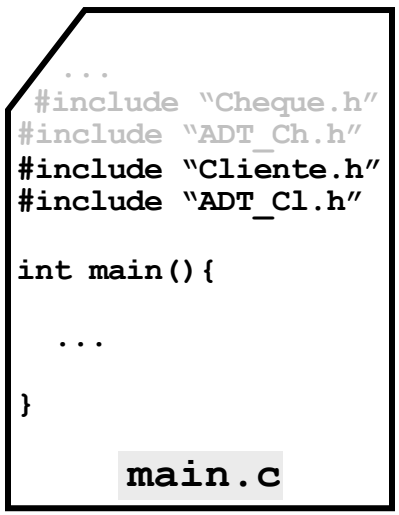
- `Cliente.h/.c` declara e implementa as funcionalidades do "Item" `Cliente`
- `ADT_Cl.h/.c` declara e implemnta as funcionalidades do ADT que organiza `Cliente`
- `main.c` pode também utilizar as funcionalidades disponibilizadas em `Cliente.h` e `ADT_Cl.h`

Sugestão de estrutura (II)





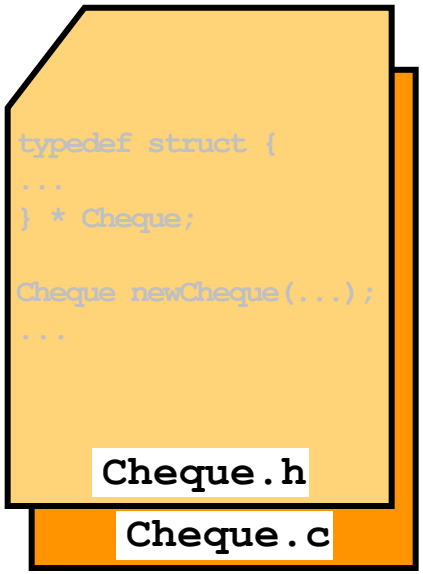


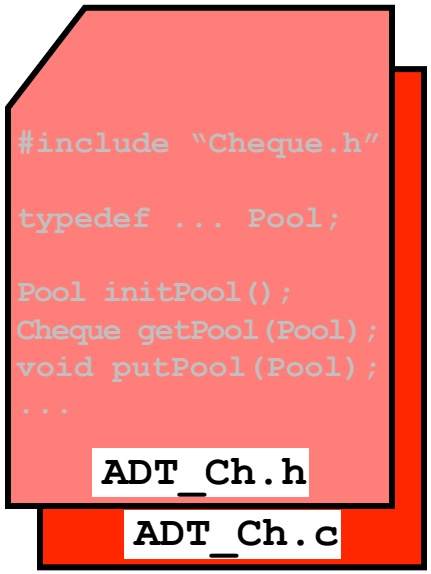


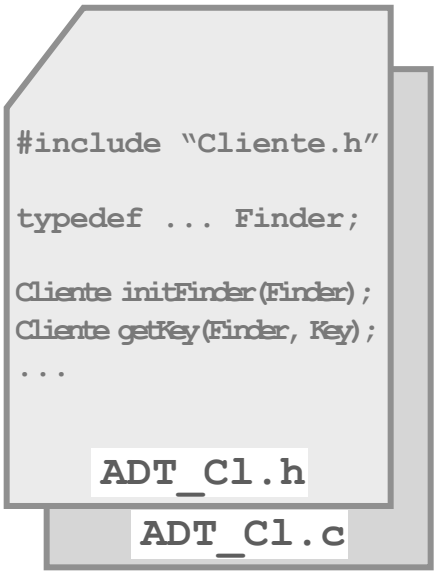
Entidade adicional - cliente:

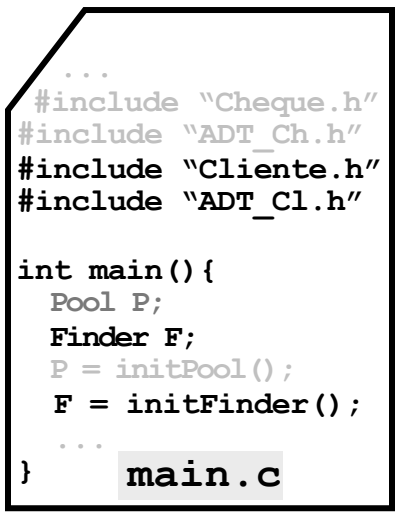
- `Cliente.h/.c` declara e implementa as funcionalidades do "Item" `Cliente`
- `ADT_Cl.h/.c` declara e implemnta as funcionalidades do ADT que organiza `Cliente`
- `main.c` pode também utilizar as funcionalidades disponibilizadas em `Cliente.h` e `ADT_Cl.h`

Sugestão de estrutura (II)





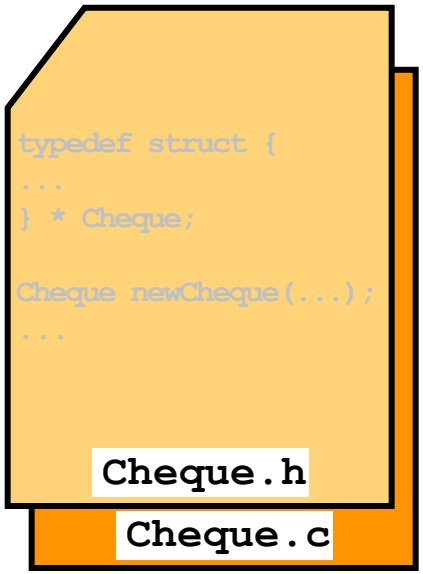


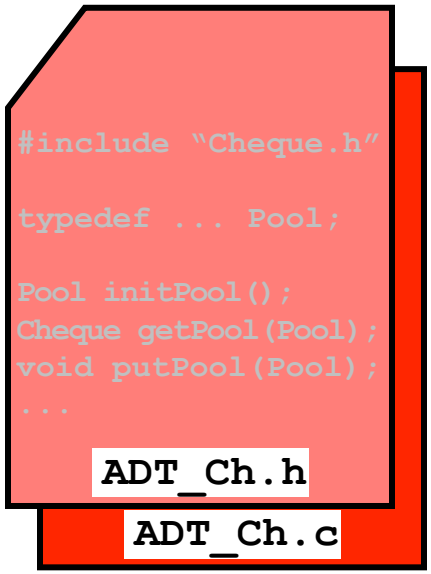


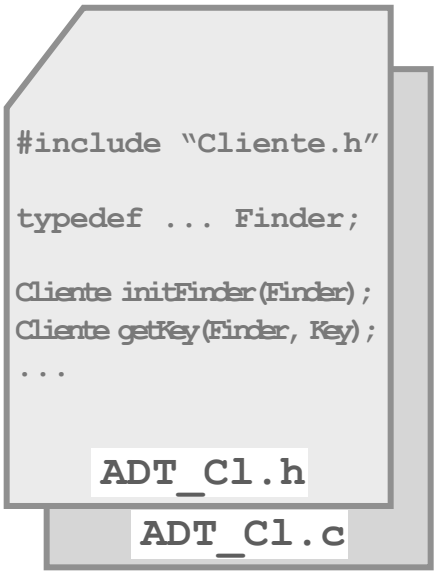
Entidade adicional - cliente:

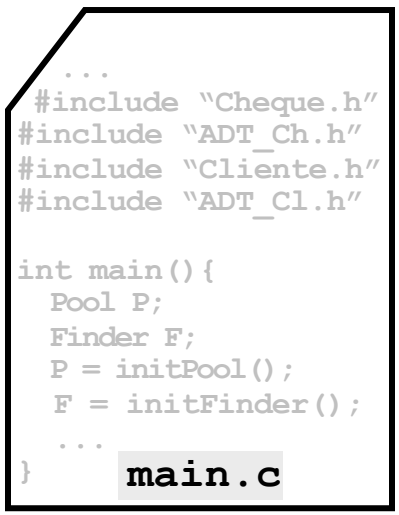
- `Cliente.h/.c` declara e implementa as funcionalidades do "Item" `Cliente`
- `ADT_Cl.h/.c` declara e implemnta as funcionalidades do ADT que organiza `Cliente`
- `main.c` pode também utilizar as funcionalidades disponibilizadas em `Cliente.h` e `ADT_Cl.h`

Sugestão de estrutura (II)



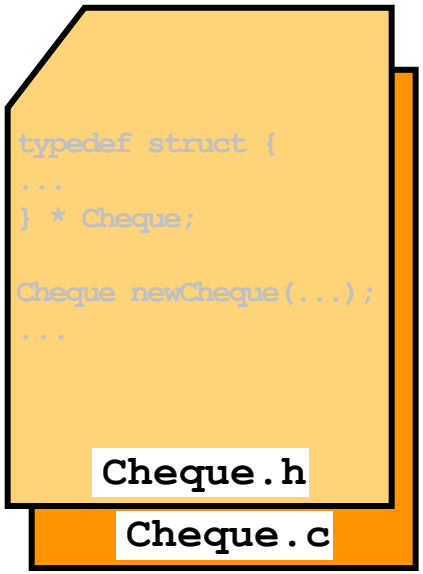


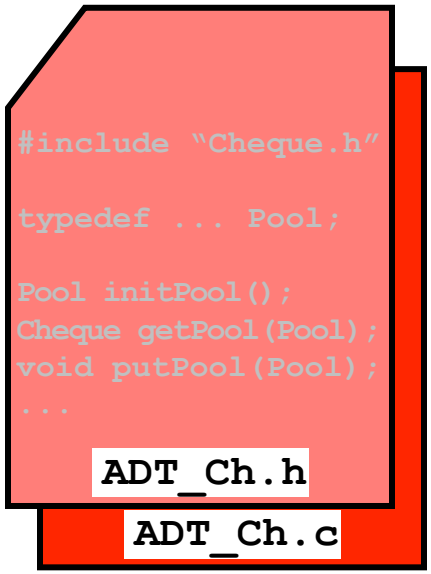


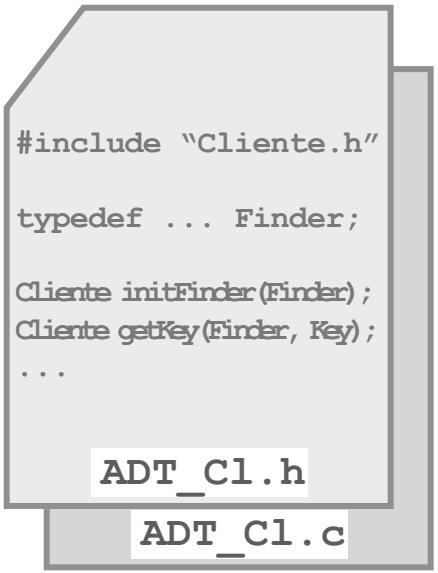


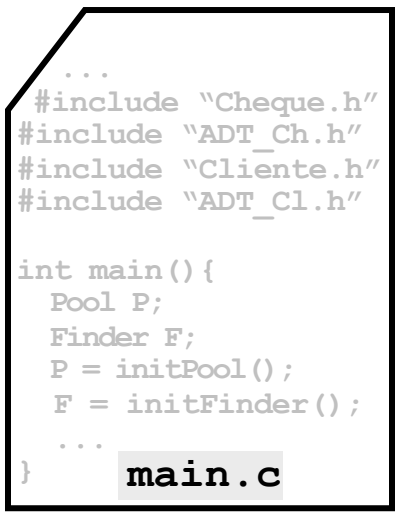

```
$ gcc -ansi -pedantic -Wall main.c Cheque.c ADT_Ch.c Cliente.c ADT_Cl.c
```

Sugestão de estrutura (III)










```
$ gcc -ansi -pedantic -Wall main.c Cheque.c ADT_Ch.c Cliente.c ADT_Cl.c
```

Sugestão de estrutura (III) (a.k.a. "*Mixórdia Informática*")

- As estruturas I e II sugeridas optimizam abstracção e modularidade...
- ... mas podem escrever-se de forma equivalente num único ficheiro.
- Disclaimer: na avaliação do projecto, correcção vale mais do que organização!

```
typedef struct {...} * Cheque;
typedef ... Pool;
typedef struct{...} * Cliente;
typedef ... Finder;

Pool initPool();
Cheque getPool(Pool);
Cheque newCheque(...);
Cliente initFinder(Finder);
Cliente getKey(Finder, Key);
...

int main(){
  ...
}

Cheque newCheque(...){
  ...
}
Pool initPool(){
  ...
}
Cheque getPool(Pool P){
  ...
}
...
```

main.c

```
$ gcc -ansi -pedantic -Wall main.c
```