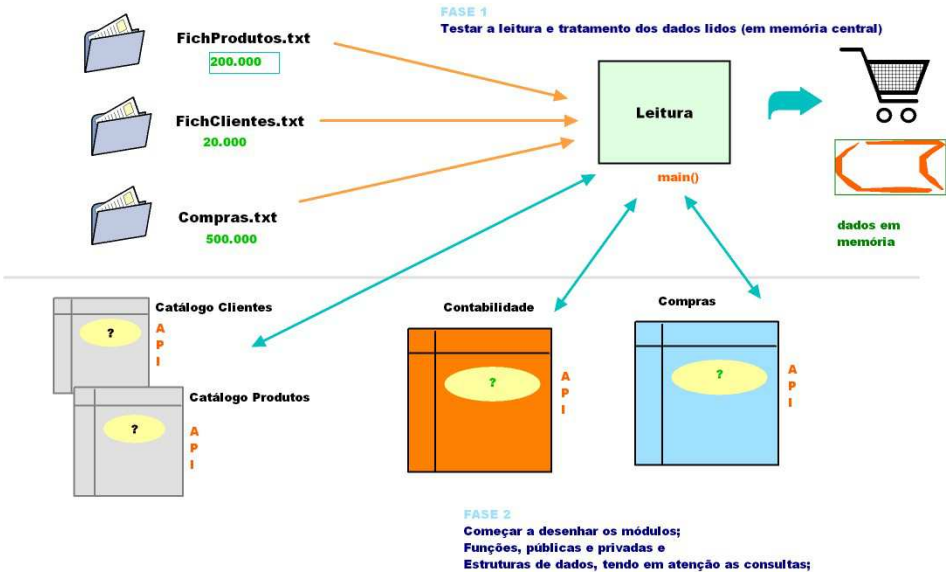
RELATÓRIO DE LI3: SEMANA 1



Tendo por base a figura acima, que desenhei no quadro, aconselhei os alunos, quer os menos experientes quer os que possam estar em fases mais avançadas do projecto, a realizarem uma FASE 1 em que os ficheiros são lidos e os dados carregados em memória, por exemplo usando arrays de strings para os códigos e, para as compras, usando um array de struct compra, por exemplo, e a realizarem testes sobre a correcção dos dados lidos.

Os alunos foram aconselhados a usar apenas funções standard, por exemplo fetgets(_,_,_) para a leitura das linhas dos ficheiros de texto, tendo-os prevenido para o facto de que a string resultado vem com \n e \0 no fim, e que portanto devem ter cuidado e realizar as necessárias operações para que depois possuam uma string com o valor correcto.

Aconselhei também a fazerem algum profiling sobre o ficheiro Compras.txt para melhor dimensionarem o tamanho do buffer de fgets().

Quanto à separação dos campos de cada linha do ficheiro Compras.txt, chamei a atenção para a existência de funções standard como strtok(), strsep(), atoi() e atof() e para a necessidade de criarem uma função trim() que não existe em C.

Cada registo de compra lido deverá ser validado segundo as regras: ambos os códigos existem, o número de unidades e o preço devem ser positivos e o mês entre 1 e 12 (de facto são todos válidos, mas ...).

Aconselhei a que, desde já, devem começar-se a medir tempos de execução e compará-los. Para tal C tem a biblioteca time.h que exporta o tipo time_t e as funções time() e difftime().

A ideia fundamental foi a de que antes de se passar à codificação dos módulos, deverá existir a garantia de que todos os dados que irão ser inseridos nos módulos após leitura a partir dos ficheiros são convenientemente tratados e estão absolutamente correctos.

Em seguida surgiram algumas questões relacionadas com a FASE 2. Aconselhei os alunos a pensarem em primeiro lugar na API, ou seja, que funções é que cada módulo necessita,

tendo dito que não se devem remeter apenas a criar as funções pedidas pelo problema, mas criar APIs o mais completas possível. Por exemplo, sendo certo que cada módulo deverá ter uma função init(...) e uma função insereX(...), qualquer que venha a ser a sintaxe, ainda que tal não seja pedido, deverão, por exemplo, implementar uma função remove(...) mesmo que nunca venha a ser usada neste problema concreto. A criação destas APIs está muito dependente do que se pretende realizar em cada consulta e sobre que módulos.

Foi chamada a atenção dos alunos para o facto deque estes módulos de dados que vão ser criados, vão ser usados no programa principal como autênticos novos tipos de dados em C, pelo que no programa principal iremos ter certamente declarações e instruções do género:

```
Contabilidade ct1, ct2;  /* se necessitássemos de ter duas contabilidades */
initContabilidade(ct1);  initContabilidade(ct2);  ou, em função da implementação,
ct1 = initContabilidade(); ct2 = initContabilidade();
```

Alguns alunos ainda não compreendem muito bem porque não se deve fazer I/O dentro do código computacional, pelo que fui muito claro com todos eles. Quem fizer I/O (no sentido teclado-monitor) nas funções dos módulos de dados C ou nas classes de Java estará imediatamente reprovado pois não respeita o Princípio da Separação.

Alguns alunos já começaram a trabalhar com os módulos de dados, mas verifiquei que nem todos leram o texto sobre tipos opacos (tipos incompletos). É necessário que sigam rigorosamente tal técnica, mesmo quando importam bibliotecas que não usam tipos incompletos!

Não há ainda muitas questões sobre os módulos Contabilidade e Compras. Mas já há algumas sobre os módulos Catálogos.

Foi também chamada a atenção para o facto de que devem desde já começar a pensar no esquema que terão que implementar no programa principal para realizarem a navegação sobre as estruturas de dados gigantescas que serão devolvidas pelos módulos a algumas das consultas. Por exemplo, se da consulta dos códigos dos Produtos iniciados pela letra A resultar uma estrutura com 8.000 códigos que é devolvida ao programa principal, como é que este irá apresentat tais dados ao utilizador, sabendo-se que cada página em ecrã terá cerca de 20 linhas disponíveis. Assim, falou-se na necessidade em certos casos de saber fazer a linearização de estruturas não lineares (cf. travessias, etc.) e, portanto, da importância do estudo destes mecanismos de navegação.

Finalmente, os alunos foram aconselhados a, para satisfazerem as regras de codificação em C ANSI standard, usarem os switches: ansi, pedantic, Wall e o2. Informei-os de que podem colocar estas directivas em CFLAGS da makefile final.

Estas foram as principais orientações resultantes das aulas práticas da 1ª semana de LI3.

F. Mário Martins