AULAS 5 E 6 - TIPOS DE DADOS ABSTRATOS

Para compreender a implementação de tipos de dados abstratos na linguagem C utilizando o paradigma modular, deve começar por ler o capítulo 2 do livro *Estruturas de Dados e Algoritmos em C*, analisando a implementação do tipo de dados abstrato COMPLEX (ponto 2.7 − Exemplo de um tipo de dados elementar matemático, páginas 50-57).

Pretende-se desenvolver o tipo de dados abstrato FRACTION para processar frações. Este tipo de dados abstrato é constituído pelo ficheiro de interface **fraction.h** e pelo ficheiro de implementação **fraction.c** (esboço da implementação) e implementa a criação de frações na forma "*numerador/denominador*" e operações sobre elas. Este tipo de dados deve ter capacidade de múltipla instanciação e usar um controlo centralizado de erros. Para armazenar as componentes de uma fração é usada uma *struct* (estrutura ou registo), com dois campos inteiros para armazenar o numerador e o denominador.

- Comece por ler com atenção os ficheiros indicados. De seguida compile o módulo, usando para o efeito o comando **gcc -c –Wall fraction.c**. A compilação deve gerar o ficheiro objeto **fraction.o**. Tente perceber o significado dos avisos indicados no monitor.

- Implemente o construtor da fração tendo em atenção que o denominador tem de ser um número inteiro positivo (no caso de uma fração negativa, o sinal está associado ao numerador), e a fração deve ser sempre armazenada na sua forma mais reduzida (ou seja, 4/6 é igual a 2/3). Para melhor estruturar a solução use as funções auxiliares `CreateFraction` e `ReduceFraction`.

- Tenha em atenção que também tem que implementar os observadores do numerador e do denominador para que o programa de simulação possa apresentar no monitor as frações criadas.

- De seguida compile a aplicação **simfraction.c** que simula as principais operações do módulo. Não se esqueça que, para compilar a aplicação, tem que mencionar o ficheiro objeto do tipo de dados (fraction.o) usando para o efeito o comando **gcc –Wall simfraction.c fraction.o –o sfrac**.

- Experimente criar as frações 2/4 e -2/-4 (a fração armazenada deve ser ½), 2/-4 e -2/4 (a fração armazenada deve ser -½). Experimente a criação de mais frações.

Para acelerar a simulação da funcionalidade deste tipo de dados também é fornecida a *makefile* **mkfraction**, para compilar o módulo e a aplicação. Deve começar por mudar-lhe o nome para **makefile** usando para o efeito o comando **mv mkfraction makefile**.

- Implemente o destrutor da fração tendo em atenção que após a libertação da memória dinâmica ela não deve ser mais acessível. Recompile o módulo e a aplicação e teste o destrutor. Se ele estiver corretamente implementado as frações apagadas do simulador não devem ser mais visíveis.

- Implemente e teste convenientemente toda a restante funcionalidade deste tipo de dados. Comece por exemplo por implementar o construtor de cópia.

- Antes de implementar as operações aritméticas básicas (adição, subtração, multiplicação e divisão) pense nas duas seguintes situações. Em qual (ou quais) destas operações há a necessidade de eventualmente ser preciso reduzir a fração? Em qual (ou quais) destas operações há a necessidade de testar o sinal do denominador para assegurar que ele obedece à condição de ser sempre positivo?

NOME: **Nº MEC:**

Pretende-se desenvolver um **tipo de dados abstrato** para processar **frações egípcias**. Uma fração egípcia é uma fração que se exprime através da soma de uma sequência de frações unitárias, distintas e por valor decrescente. Por exemplo:

$$\frac{3}{4}=\frac{1}{2}+\frac{1}{4} \qquad \frac{7}{15}=\frac{1}{3}+\frac{1}{8}+\frac{1}{120} \qquad \frac{8}{11}=\frac{1}{2}+\frac{1}{5}+\frac{1}{37}+\frac{1}{4070}$$

O processo de conversão de uma fração própria positiva para uma fração egípcia é dado pelo seguinte algoritmo ganancioso que determina a próxima fração unitária:

$$\frac{x}{y}=\frac{1}{\lceil y/x \rceil}+\frac{(-y)\ mod\ x}{y \times \lceil y/x \rceil}$$

Este algoritmo de conversão é aplicado pela função interna iterativa **CreateUnitFraction**, que tem como parâmetro de entrada uma fração (que se assume uma fração própria positiva) e que devolve a maior fração unitária existente ou uma referência nula no caso de falta de memória dinâmica para alocar uma fração. A função altera a fração passada no parâmetro de entrada para o valor do resto da fração que precisa de continuar a ser convertida. Caso haja *overflow* no cálculo do denominador desta fração restante, então o parâmetro de entrada fica com a referência nula para indicar a impossibilidade de fazer a extração da fração unitária seguinte. Por exemplo, a fração 5/121 a partir do terceiro termo já não é representável em frações unitárias (com campos de tipo ***integer***), pelo que a fração vai ficar incompleta e composta apenas pelas primeiras três frações unitárias:

$$\frac{5}{121}=\frac{1}{25}+\frac{1}{757}+\frac{1}{763309}+\frac{1}{873960180913}+\frac{1}{1527612795642093418846225}$$

Como o parâmetro de entrada fica corrompido pela execução da função, qualquer função que a utilize deve – depois das validações necessárias – fazer uma cópia da fração e usar a cópia, não se esquecendo de eliminá-la no fim. Tenha em atenção que o processo de conversão é aplicado até a fração restante ser a fração nula, até se atingir *overflow* ou até se exceder a capacidade de armazenamento da estrutura de dados de suporte (caso ela seja estática).

Pretende-se completar duas implementações baseadas em duas estruturas de dados diferentes:

- Na implementação **estática** as frações são armazenadas num *array*. Como estamos perante uma estrutura de dados estática vamos limitar a capacidade de armazenamento a 10 frações unitárias. A funcionalidade pretendida é especificada pelo ficheiro de interface **egyptianfraction.h**, sendo também fornecido o esqueleto do ficheiro de implementação **arrayegyptianfraction.c**.

- Na implementação **dinâmica** as frações são armazenadas numa lista ligada. Como estamos perante uma estrutura de dados dinâmica ela vai crescendo em função das necessidades. A funcionalidade pretendida é especificada pelo ficheiro de interface **egyptianfraction.h**, sendo também fornecido o esqueleto do ficheiro de implementação **listegyptianfraction.c**.

Também é fornecido o programa de simulação **simegyptian.c** e a *makefile* **mkegyptian** para compilar o projeto.