Aula 12

Objectivo: Excepções e problemas finais

Problema 1

Em geral sempre que se pretende o desenvolvimento de programas tolerantes a falhas (ou seja, programas que mesmo na presença de [certos] erros, continuam a dar resultados correctos), faz-se uso da redundância de código. Neste problema pretende-se a implementação de uma estrutura simples de tolerância a erros no cálculo da função `seno(x)`, fazendo uso (na ordem apresentada) dos seguintes algoritmos:

1. $seno\_1(x) = x\ , \qquad erro \leq \left|x^3/3!\right|$
2. $seno\_2(x) = x - x^3/3!\ , \qquad erro \leq \left|x^5/5!\right|$
3. $seno\_3(x) = \sum_{i=0}^{3} \frac{(-1)^i * x^{2*i+1}}{(2*i+1)!}\ , \qquad erro \leq \left|x^9/9!\right|$

(Estes algoritmos assentam no desenvolvimento em série de Maclaurin para a função $seno(x)$)

1.1. Construa um módulo de cálculo numérico de funções matemáticas, que permita o cálculo numérico da função $seno(x)$. Este módulo deve permitir que se defina um erro (absoluto) máximo aceitável no cálculo, e o cálculo do *seno* deve ser feito por tentativas (fazendo uso do mecanismo de excepções do `C++`), desde o algoritmo 1, até que o erro seja aceitável ou então até ao algoritmo 3. Caso nenhum algoritmos dê um resultado aceitável o módulo deve propagar a excepção para os seus clientes (mas apenas nesta situação).

1.2. Faça com que as excepções geradas levem o valor o erro máximo da estimativa de cada algoritmo.

Problema 2

Uma das técnicas que na última década mais tem contribuído para a correcção (capacidade de um programa funcionar de acordo com a sua especificação) do software, é a chamada programação por contrato. Esta técnica consiste, essencialmente, na associação de asserções (condições) às várias

entidades dos programas (asserções são condições que têm sempre que se verificar no sitio onde são aplicadas).
Há vários tipos possíveis de asserções: pré-condições – condições a verificar na chamada de rotinas (funções, procedimentos, ou métodos de classes), pós-condições (a verificar no fim da execução de rotinas), invariantes (propriedades das classes), e asserções de verificação noutros pontos de programas.
Um exemplo simples de pré-condições e pós-condições são as que se aplicam à função `sqrt`. Esta função só pode ser utilizada sse o seu argumento for um número não negativo:

```
double sqrt(double n)
{
  requer(n >= 0);

  double result;

  (...)

  garante(result * result == n);

  return result;
}
```

(NOTA: Até agora temos usado o `assert` para este fim)
Um dos problemas levantados com o uso de asserções, é como deve o programa reagir quando alguma destas não se verificar. Sendo verdade que na grande maioria das situações o comportamento adequado será terminar de imediato o programa relatando e identificando o ponto onde a asserção falhou (especialmente durante o desenvolvimento do programa), há certas situações (tolerância a falhas) onde tal não pode ocorrer (por exemplo num sistema de piloto automático de um avião). As excepções do `C++` permitem-nos a possibilidade de adaptar o comportamento das asserções às nossas necessidades (garantindo, por defeito, que o comportamento é o de terminar o programa).
Neste problema pretende-se implementar as seguintes asserções, dando a possibilidade de se detectar a ocorrência de uma qualquer destas falhas de asserções (ou uma qualquer delas), fazendo uso das excepções do `C++`:

- pré-condições: `requer`;
- pós-condições: `garante`;

- verificações: `verifica`.

(Os invariantes não são facilmente implementáveis em `C++`, pelo que serão omitidos)
NOTA: Para dar a possibilidade de detecção de uma qualquer destas asserções, podem-se criar hierarquias de classes de excepções.

2.1. Aplique estas asserções nas classes `ARRAY` desenvolvidas nas duas aulas anteriores.

Problema 3

Projecte e construa um conjunto de módulos que permitam a codificação e descodificação de mensagens (de texto), de forma a que estes sejam facilmente extensíveis para novos tipos de codificadores.
Numa primeira versão implemente apenas um codificador que vire a mensagem ao contrário (`"batata"` passa a `"atatab"`).
Sugere-se que este problema seja abordado da seguinte forma:

1. Identifique as classes (módulos) que fazem sentido criar, tentando que estas sejam tão abstractas (genéricas) quanto possível;
2. Construa (testando) essas classes;
3. Faça um programa que permita a codificação, envio, recepção e descodificação de mensagens (atenção que o emissor e o receptor apenas interagem através do **texto** da mensagem codificada).

3.1. Acrescente os seguintes codificadores:

- XOR (operador `^` do `C++`);
- Converta as letras minúsculas para maiúsculas, rodando duas posições para a frente (`"a"` passa a `"C"`, `"x"` a `"Z"`, `"y"` a `"A"`, etc.).