Universidade Federal de Minas Gerais
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
Disciplina: Algoritmos e Estruturas de Dados II (DCC004)
Professor: Renato Martins (renato.martins AT dcc.ufmg.br)
https://www.dcc.ufmg.br/~renato.martins/courses/DCC004
2° Semestre de 2018

Lista de Revisão II – Refatoração, Programação Defensiva, Tratamento de Exceções e Testes

1. O que é serialização e desserialização de objetos?
2. O que são asserções? Qual a diferença entre asserções e exceções em C++?
3. Como lançamos exceções em C++? Como tratamos exceções? (usar os conceitos envolvidos com **try, catch** e **throw**)
4. O que é cobertura de código (ou cobertura de teste)?
5. Qual a diferença entre as técnicas de teste de código `White-Box` e `Black-Box`?
6. O que é a técnica de desenvolvimento de código TDD (**Test Driven Development**)?
7. Defina e comente as diferenças entre os diferentes níveis de teste de código: teste de unidade, teste de integração e teste de sistema.
8. Qual a saída do seguinte programa?

```
#include <iostream>
int main()
{
   int x = -1;
   try {
      std::cout << "Dentro do try " << std::endl;
      if (x < 0)
      {
         throw x;
         std::cout << "Depois do throw " << std::endl;
      }
   }
   catch (int x ) {
      std::cout << "Excecao tratada " << std::endl;
   }

   std::cout << "Depois de catch "<< std::endl;
   return 0;
}
```

9. A saída do seguinte programa:

```
#include <iostream>

class Base {};
class Derivada: public Base {};
int main()
{
    Derivada d;
    try {
        throw d;
    }
    catch(Base &b) {
         std::cout << "Peguei excecao pela classe base" << std::endl
             ;
    }
    catch(Derivada &d) {
         std::cout << "Peguei excecao pela classe derivada" << std::
             endl;
    }
    return 0;
}
```

é: "Peguei excecao pela classe base". Modifique minimamente o código (sem acrescentar nenhuma instrução) de forma que ele retorne: "Peguei excecao pela classe derivada".

10. A leitura e escrita em posições fora das dimensões de um array resulta em um "comportamento indefinido" em C++ [1]. Para prevenir esse tipo de situação, a `std::vector` da STL fornece o operador de acesso `at()` que verifica se a posição é válida. Caso não seja válida ela lançará uma exceção, como no exemplo a seguir:

```
#include <iostream>
#include <vector>
#include <stdexcept>

int main(){
    std::vector<int> vetor(10);
    try{
       for(int i = 0; i < 15; i++) {
            vetor.at(i) = i;
         }
    }catch (std::exception &e) {
       std::cout << "Posicao invalida: " << e.what() << std::endl;
    }
    try{
      std::cout << "Valor na posicao 15: " << vetor.at(15) << std::
         endl;
    }catch(std::exception &e){
      std::cout << "Posicao invalida: " << e.what() << std::endl;
    }
    std::cout << "Fim do programa" << std::endl;
    return 0;
}
```

[1] https://en.wikipedia.org/wiki/Undefined_behavior

Terá como resultado:

```
Posicao invalida: vector::_M_range_check: __n (which is 10) >= this->size() (which is 10)
Posicao invalida: vector::_M_range_check: __n (which is 15) >= this->size() (which is 10)
Fim do programa
```

No entanto, se usarmos o operador [ ], a exceção não será lançada e o programa terá comportamento não definido. Queremos "corrigir" esse tipo de situação, com a criação de um novo tipo de array de inteiros bem simples (classe `MeuArray`), que tenha o operador [ ] com um comportamento semelhante ao método `at()`. Complete o código do operador [ ] na classe `MeuArray` de forma a ter o comportamento esperado no código a seguir.

```
#include <iostream>
#include <stdexcept>

class MinhaExcecao : public std::exception {
 private:
    std::string _message;
 public:
   MinhaExcecao(int dimensao, int posicao) {
    _message =  "range_check: __n (which is " + std::to_string(
       dimensao) + " ) >= this->size() (which is " + std::to_string
       (posicao) + " )";
   }
   const char* what() const throw() {
        return _message.c_str();
    }
};

class MeuArray{
 public:
  int _tamanho;
  int *_dado;
  MeuArray(int tamanho){
    _tamanho = new int[tamanho];
    this->_tamanho = tamanho;
  }

  int &operator[](int posicao) const{
  // Complete este metodo
   }
 };

int main(){
    MeuArray vetor(10);
    try{
       for(int i = 0; i < 15; i++) {
            vetor[i] = i;
         }
    }catch (std::exception &e) {
       std::cout << "Posicao invalida: " << e.what() << std::endl;
    }
    try{
      std::cout << "Valor na posicao 15: " << vetor[15] << std::
         endl;
    }catch(std::exception &e){
```

```
42        std::cout << "Posicao invalida: " << e.what() << std::endl;
43    }
44    std::cout << "Fim do programa" << std::endl;
45    return 0;
46 }
```

11. Encontre e discuta brevemente como resolver pelo menos dois "Bad smells" no nosso TAD `MeuArray` do exercício anterior.

12. Ainda em relação a questão **10**, implemente uma segunda classe derivada de `std::exception`, `MinhaExcessao2` que mostre a mensagem "`Minha excecao ao checar o tipo de posicao, nao e do tipo unsigned int`" caso o valor de posição não seja uma dimensão positiva. Modifique o operador [ ] de forma que o seguinte código principal:

```
1  int main(){
2      MeuArray vetor(10);
3      try{
4          for(int i = 0; i < 15; i++) {
5              vetor[i] = i;
6          }
7      }catch (std::exception &e) {
8          std::cout << "Posicao invalida: " << e.what() << std::endl;
9      }
10     try{
11       std::cout << "Valor na posicao -1: " << vetor[-1] << std::
             endl;
12     }catch(std::exception &e){
13       std::cout << "Posicao invalida: " << e.what() << std::endl;
14     }
15     std::cout << "Fim do programa" << std::endl;
16     return 0;
17 }
```

tenha como resultado:

```
Posicao invalida:  vector::_M_range_check:  __n (which is 10) >= this- >size() (which is 10)
Posicao invalida:  Minha excecao ao checar o tipo de posicao, nao e do tipo unsigned int
Fim do programa
```

13. Queremos definir uma função que calcula o fatorial de um número. Essa função tem como requisito as seguintes condições de entrada e saída:

- meuFatorial(0): retorna o inteiro 1
- meuFatorial(-1): lança uma exceção que imprime a seguinte mensagem "Entrada nao valida"
- meuFatorial(3): retorna o inteiro 6

(a) Implemente a função `meuFatorial` baseada nesses requisitos.

(b) Utilizando a biblioteca `doctest.h`, crie um `main_teste.cpp` que faça testes de unidade para os resultados da sua implementação.