Universidade Federal de Minas Gerais
departamento de ciência da computação
Disciplina: Algoritmos e Estrutura 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 1 – Revisão de Conceitos Básicos (Sudoku++) e Programação Orientada a Objetos

Data de publicação: 31/08/2018
Data de Entrega: 28/09/2018 (via email e com demonstração em aula)

Importante: Esta lista de exercícios contém duas partes P_1 e P_2 (cada uma valendo 10 pontos). A pontuação de cada item é indicada entre colchetes antes de cada questão. A nota final da lista ($NL \in [0,\ 10]$) é dada por:

$$NL = \begin{cases} \left(\frac{P_1+P_2}{2}\right), \text{ se } P_1 \geq 5 \text{ e } P_2 \geq 5 \\ \min(P_1, P_2), \text{ caso contrário.} \end{cases}$$

onde min fornece o valor mínimo entre P_1 e P_2. A primeira parte da lista é de revisão de conceitos de Algoritmos e Estruturas de Dados I, onde vocês farão um solver de sudoku. A segunda parte é sobre a modelagem de diferentes problemas usando programação orientada a objetos. Vocês terão de enviar suas soluções em C++ via email até a data de entrega e os programas deverão ser mostrados em aula no dia da entrega.

Parte I – Revisão de Conceitos Básicos (Sudoku++)

O Sudoku é um quebra-cabeça japonês bastante popular. Em sua versão mais tradicional, o objetivo é o preenchimento de um diagrama 9×9 obedecendo as seguintes restrições:

- As linhas devem conter números de 1 a 9 sem repetições.
- As colunas devem conter números de 1 a 9 sem repetições.
- O diagrama é dividido em 9 regiões 3×3 e estas também devem conter números de 1 a 9 sem repetições.

Observe que o diagrama pode ter dimensão N×N diferente de $N = 9$ (veja alguns exemplos aqui: https://en.wikipedia.org/wiki/Sudoku#Variants). Se você ainda não jogou a versão tradicional (9×9), tente um site como websudoku ou http://pt.sudokupuzzle.org/, de onde foi retirado o exemplo mostrado na Figura 1. Nesta primeira parte da lista, você deverá escrever um programa em C++ que resolva um Sudoku de tamanho 9×9 e em seguida irá generalizar o seu algoritmo para resolver um Sudoku de tamanho genérico N×N, em que $N = K^2$ e K inteiro.

	5			6				
	9						4	7
3			1				8	5
			5	1	9			
8	1							
				2		7	9	
	2	6		3		1		
			6				7	
	4	1					6	

Figura 1: Sudoku nível de dificuldade "super". Imagem cortesia de http://pt.sudokupuzzle.org/.

Estrutura do Projeto

O seu projeto deverá ser compilado usando Make (https://en.wikipedia.org/wiki/Make_(software)) e deverá conter, pelo menos, um Makefile, um arquivo `main_sudoku.cpp` contento a chamada principal, e os arquivos `lib/tools.h` e `lib/tools_impl.h` contendo as definições e implementações das funções auxiliares.

1. **[2pt] Leitura da entrada do Sudoku (grid)**
   Implemente rotinas que realizem a leitura de um diagrama Sudoku fornecido via um arquivo de entrada. A codificação do arquivo poderá ser tanto em formato texto (ascii) ou binário (uint32_t), como mostrado nos dois arquivos fornecidos em `https://www.dcc.ufmg.br/~renato.martins/courses/DCC004/lists/input_sudoku.tar.gz`. Vocês podem verificar o formato do arquivo `input_sudoku_ascii` com um editor de texto ou com o comando `less input_sudoku_ascii` no terminal. O arquivo binário (`input_sudoku_bin`) contém o diagrama na forma de uma matriz 9×9, com inteiros entre $[0,\ 9]$, em que as posições da matriz com valor 0 correspondem às células não preenchidas do Sudoku.

2. **[1pt] Impressão no terminal**
   Implemente uma rotina que mostre no terminal do usuário o grid a ser resolvido, conservando o mesmo estilo do diagrama no arquivo de entrada no formato texto (`input_sudoku_ascii`).

3. **[5pt] Solução para o caso $N = 9$**
   Implemente as rotinas que completem o sudoku para o caso $N = 9$. O diagrama sendo completado a cada iteração deve ser mostrado no terminal. Caso o sudoku não tenha solução, o seu programa deverá mostrar a mensagem "Sudoku sem solução".

4. **[2pt] Sudoku para dimensões superiores $N = 16$ e $N = 25$**
   Crie um outro projeto e adapte o programa anterior de forma que o seu algoritmo seja capaz de resolver Sudokus de diferentes tamanhos (sem grandes alterações no código). O diagrama sendo completado a cada iteração deve ser mostrado no terminal.

PARTE II – Programação Orientada a Objetos e Classes

Instruções: Para cada um dos exercícios a seguir, crie um projeto como detalhado na **Parte I** desta lista. Relembrando: o projeto deverá ser compilado usando Make (conter um Makefile), ter pelo menos um arquivo `.h` que defina os atributos e métodos da classe (contrato da classe), um arquivo `.cpp` que implemente os métodos da classe (comportamento) e um arquivo principal `.cpp` que instancie um objeto da classe que você definiu. Não se esqueça de usar "guarda de segurança"(`#ifndef #define #endif`) nos seus cabeçalhos (headers `.h`).

5. [**2pt**] Crie uma classe `Data` com três atributos inteiros: `dia`, `mês` e `ano`. Faça um construtor que inicializa as três variáveis e suponha que os valores passados serão corretos. A classe deve possuir um método para exibir a data em formato de números separados por barra: dia/mes/ano e outro método para exibir a data por extenso (ex: 12 de janeiro de 2015).

6. [**2pt**] Crie uma classe `Rectangle` com atributos `length` e `width`, cada um dos quais assume o padrão de 1. Forneça funções-membro que calculam os atributos `perimeter` e `area` do retângulo. Além disso, forneça as funções `set` e `get` para os atributos `length` e `width`. As funções `set` devem verificar se `length` e `width` são números de ponto flutuante maiores que $0,0$ e menores que $20,0$.

7. [**2pt**] Implemente em C++ uma classe chamada `Aquecedor`. Ela deve ter um único atributo chamado `temperatura`, cujo tipo deve ser um ponto flutuante de precisão dupla. Defina um construtor que não recebe parâmetros e inicializa a temperatura em 20 graus. Crie os métodos aquecer e resfriar que aumentam e diminuem a temperatura em 5 graus, respectivamente. Defina um método para retornar o valor da temperatura.

8. [**2pt**] Altere a classe do exercício anterior para que ela tenha três novos atributos: temperatura mínima, temperatura máxima e fator de incremento da temperatura. Os dois primeiros devem ser inicializados com 10 e 40 graus respectivamente no construtor. A classe deve ter um construtor sem parâmetros, que definirá o fator de incremento em 5 graus, um segundo construtor que recebe a temperatura inicial e um terceiro que recebe a temperatura inicial e o fator de incremento.

Altere os métodos existentes na classe de forma apropriada com o objetivo de manter o estado do objeto sempre válido (ex: o fator de incremento deve ser usado toda vez que os métodos aquecer e resfriar forem chamados). Escreva mensagens na saída padrão quando uma ação não puder ser executada por não ser um estado de objeto válido.

Por fim, crie um método que permita alterar o fator de incremento da temperatura depois de um objeto já ter sido criado.

9. [**2pt**] Crie uma classe `SavingsAccount`. Utilize um membro de dados `static annualInterestRate` para armazenar a taxa de juros anual para cada um dos correntistas. Cada membro da classe contém um membro de dados `private savingsBalance` para indicar a quantia que os correntistas têm atualmente em

depósito. Forneça a função-membro `calculateMonthlyInterest` que calcula os juros mensais multiplicando o `balance` [saldo] pelo `annualInterestRate` dividido por 12; esses juros devem ser adicionados a `savingsBalance`. Forneça uma função-membro `static modifyInterestRate` que configura o `static annualInterestRate` com um novo valor. Escreva um programa de driver para testar a classe `SavingsAccount`. Instancie dois objetos diferentes da classe `SavingsAccount`, `saver1` e `saver2`, com saldos de $ 2.000,00 e $ 3.000,00, respectivamente. Configure o `annualInterestRate` como 3%. Em seguida, calcule os juros mensais e imprima os novos saldos de cada um dos correntistas. Então configure o `annualInterestRate` como 4%, calcule os juros do próximo mês e imprima os novos saldos para cada um dos poupadores.