Universidade Federal do Rio Grande do Sul
Instituto de Informática
INF01202 - Algoritmos e Programação - Turmas I/J - 2020/1
Prof. Renan Maffei e Prof. Lucas Schnorr

Especificação do Trabalho Final

1. Motivação e Objetivos

O trabalho final tem por objetivo exercitar as habilidades e conceitos de programação desenvolvidos ao longo da disciplina INF01202 através da **implementação de um jogo de computador em linguagem C**. Este trabalho também permitirá ao aluno exercitar as habilidades de pesquisa, uma vez que será possível utilizar bibliotecas e conceitos ainda não trabalhados em aula para implementar certas funcionalidades. Outro objetivo do trabalho é exercitar a habilidade de desenvolvimento em equipe, uma vez que o trabalho (implementação) deve ser **realizado por duplas** de alunos.

2. Requisitos Administrativos

2.1. Etapas de entrega

O trabalho será entregue em 3 etapas:

A. Andamento Data: 30/10/2020

Envio do código parcial do trabalho, acompanhado de breve relato para explicar o andamento do trabalho e mostrar o que já está feito. É obrigatório ter alguma parte do código do trabalho funcional e que possibilite uma demonstração (em video pré-gravado, duração < 5 min). O não cumprimento desta etapa implicará em desconto de 10% da nota final obtida pelo aluno!

OBS: A formalização das duplas deve ser feita e informada aos professores com no mínimo uma semana de antecedência (até 23/10/2020). Será disponibilizado um espaço no Moodle para isso.

B. Entrega Final Data: 22/11/2020

A dupla deverá submeter via Moodle um arquivo zip contendo:

- Um relatório com a descrição do trabalho realizado, a especificação completa de como os elementos do jogo foram representados, como foi implementada a interação dos componentes interativos, bem como as estruturas e funções utilizadas e uma explicação de como usar o programa.
- Os códigos fontes devidamente organizados e documentados (.c).
- Bibliotecas necessárias para executar os códigos.

C. Apresentação Data: 23/11/2020 a 25/11/2020

Os alunos devem ser capazes de explicar todos os recursos da linguagem, comandos e bibliotecas utilizados no seu código. É permitido usar recursos não vistos em aula desde que se saiba explicar como funcionam e para que servem. Ambos membros da

dupla devem saber responder perguntas sobre qualquer trecho do código e estes serão avaliados separadamente.

As apresentações ocorrerão no horário das aulas via Teams.

2.2. Avaliação

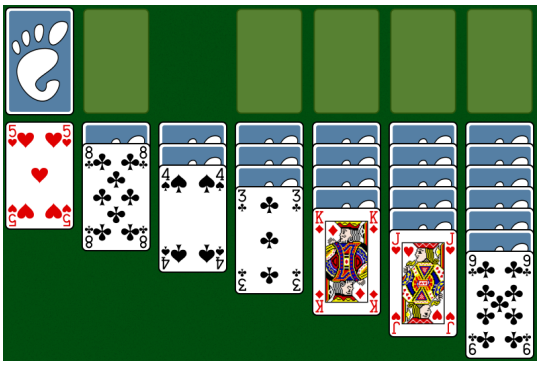
A avaliação levará em conta os seguintes critérios:

- Atendimento aos requisitos definidos;
- Estruturação do código em módulos;
- Uso das estruturas e funcionalidades vistas em aula;
- Documentação geral do código (comentários);
- Indentação do código;
- "Jogabilidade" do jogo.

Importante: Trabalhos copiados não serão considerados. Existem ferramentas que possibilitam a detecção automática de plágio, as quais serão utilizadas na correção. **Se for detectado plágio, todos os trabalhos relacionados serão desconsiderados**.

3. Descrição Geral e Regras do Jogo a Ser Implementado

O jogo que deverá ser implementado é o jogo de cartas **Paciência** (também conhecido como *Solitaire* ou *Klondike)*. Ele é um jogo de cartas jogado com um baralho completo de 52 cartas, composto por grupos de 13 cartas (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K) de quatro diferentes naipes (***paus***, *copas*, ***espadas***, *ouros*).



Conforme mostra a Figura 1, as cartas do baralho são distribuídas sobre uma mesa em 4 regiões, conhecidas como o **Tableau**, a **Fundação**, o **Estoque** e o **Descarte**. As regras de distribuição das cartas são as seguintes:

- **Tableau**: composto por 7 pilhas de cartas na parte inferior da mesa.
  - As pilhas iniciam com cartas aleatórias viradas para baixo, em quantidade crescente da esquerda para direita (ex: a pilha mais à esquerda inicia com 1 carta e a pilha mais à direita inicia com 7 cartas).
  - No inicio do jogo, a carta no topo de cada pilha é virada para cima, conforme exemplificado na Figura 1.
    OBS: na visualização do tableau, a carta do topo é a carta dentro da pilha que está desenhada mais para baixo. Diz-se que ela está no topo, pois ela está se sobrepondo as demais, e precisa ser retirada primeiro para que as outras possam ser mexidas.
  - Uma pilha pode receber uma nova carta (de outras pilhas, da fundação ou do descarte), desde que o valor da carta seja **imediatamente menor** ao da carta no topo da pilha e sua **cor seja diferente**. A nova carta passa a ser o novo topo da pilha.

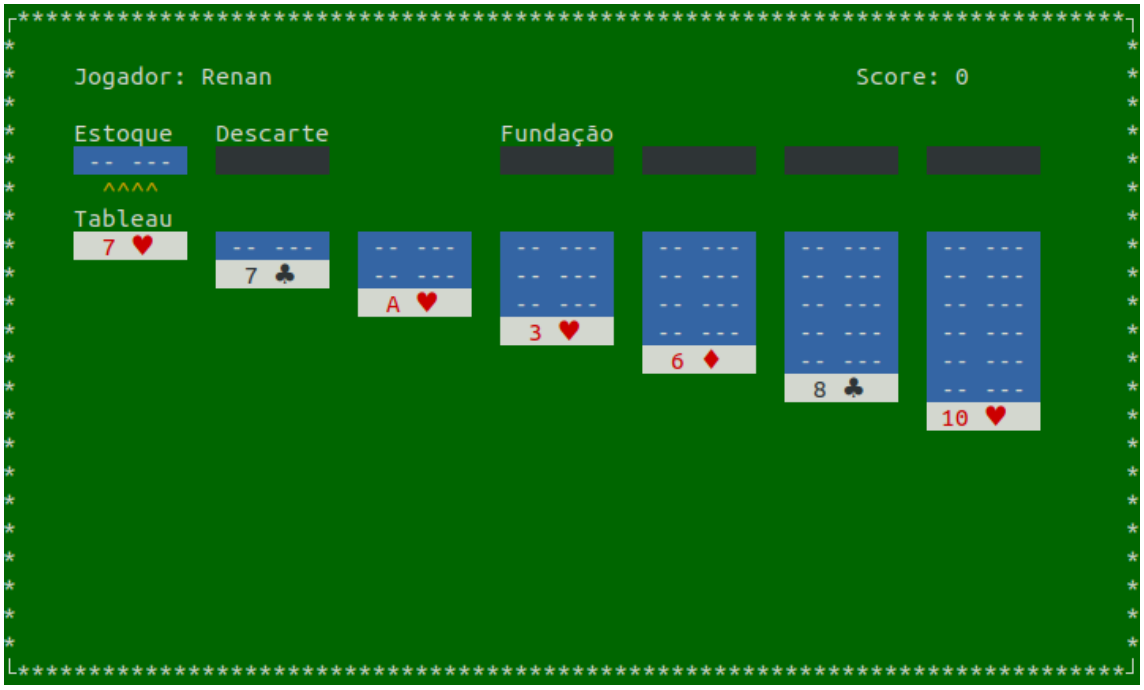


Figura 1. Exemplo de implementação do jogo em modo texto, mostrando a distribuição das cartas na mesa

  - Uma carta (ou uma sequência de cartas de valores alternados) de uma dada pilha X pode ser movida para outra pilha Y, desde que a primeira carta desta sequência tenha valor **imediatamente menor** e **cor diferente** da carta no topo da pilha Y.

    OBS: não é possível mover cartas que ainda estão viradas para baixo.
  - Pilhas vazias (sem cartas) podem receber cartas de valor Rei (K) ou sequências de cartas que comecem por um Rei (K).
  - Quando todas as cartas viradas para a cima de uma pilha são retiradas desta pilha, a próxima carta virada para baixo é desvirada.
- **Fundação**: composto por 4 pilhas de cartas na parte superior direita da mesa, sendo uma para cada naipe.
  - Cada pilha da fundação **inicia vazia** e a primeira carta que deve receber é o Ás (A) do naipe correspondente.
  - Cada pilha da fundação só pode receber a carta de valor **imediatamente superior** e **mesmo naipe** da carta do seu topo.
  - Caso o jogador ache necessário, é possível mover a carta do topo de uma dada pilha da fundação (e somente a carta do topo) para uma pilha do tableau.
- **Estoque**: uma pilha na parte superior esquerda da mesa, com as cartas restantes do baralho, que são mantidas **viradas para baixo**.
  - Inicialmente, o estoque armazena em ordem aleatória as cartas que não foram distribuídas no tableau.
  - A cada jogada, é possível retirar uma carta por vez do estoque. A carta retirada vai **automaticamente** para o topo da pilha de **descarte** virada para cima.

  - Quando o estoque ficar vazio, em uma próxima jogada ele pode receber novamente todas as cartas da pilha de descarte, que serão novamente viradas para baixo. A **ordem** das cartas **deve ser mantida**, ou seja, a carta mais ao fundo da pilha de descarte assumirá o topo do estoque, e assim sucessivamente.
- **Descarte**: pilha ao lado do estoque que recebe as cartas descartadas.
  - O descarte inicia **vazio** e a cada jogada pode receber uma carta do estoque, se o jogador assim quiser.
  - Apenas a carta do topo do descarte pode ser movida, tanto para alguma pilha da fundação, quanto para alguma pilha do tableau. O movimento deve seguir as regras especificadas anteriormente.
  - Não é possível mover uma carta do tableau ou da fundação de volta para o estoque.

Seguindo as regras especificadas acima, o **objetivo** do jogador é movimentar continuamente as cartas de uma pilha para as outras, até que todas as cartas de cada naipe terminem na sua pilha correspondente na fundação.

Paciência é um jogo de estratégia, mas também de sorte, portanto é importante ter em mente que, dependendo da configuração inicial gerada aleatoriamente na distribuição das cartas, nem sempre haverá uma solução para ganhar o jogo, ou a soluçao poderá ser muito difícil de encontrar.

3.1. Cálculo do Score do Jogo

- 5 pontos - Mover uma carta do **descarte** para o **tableau**
- 5 pontos - Desvirar uma carta do **tableau** (que estava virada para baixo)
- 10 pontos - Mover uma carta para a **fundação**
- -15 pontos - Retornar uma carta da **fundação** para o **tableau**
- -50 pontos - Reciclar as cartas do **descarte** de volta para o **estoque**
- 300 pontos - Finalizar o jogo com sucesso (todas as cartas na **fundação**)

O escore mínimo é zero.

4. Requisitos Mínimos de Implementação

A implementação realizada pelos alunos deverá respeitar os seguintes requisitos mínimos:

4.1. Visualização da tela de jogo

- Os elementos visuais do jogo devem ser implementados e exibidos visualmente. A representação visual pode ser em modo texto, como na Figura 1, mas é possível utilizar formas alternativas de representação visual.
  - **Exemplos de biblioteca para interface em modo texto:** `conio.h` (para Windows) e `ncurses.h` (para Linux, que foi usada na implementação do programa da Figura 1)
- A tela de jogo deve mostrar a mesa com as pilhas de cartas, um cursor que se movimentará pelas pilhas indicando as jogadas (em amarelo, na Figura 1), o nome do jogador e o score do jogo.

4.2. Tela de abertura e Carregamento de Configuração Salva

- No início, o programa deverá ler o **Nome do jogador** e permitir a escolha de um **Novo Jogo** ou **Carregar** uma configuração salva.
- Se for escolhido **carregar** uma configuração salva, o usuario deve informar o nome do arquivo contendo essa configuração, e o programa deverá carregá-la.
  - A configuração a ser carregada/salva é o embaralhamento das cartas feito no início do jogo. Escolher carregar uma configuração existente, começará um jogo do inicio, mas sempre com a mesma distribuição (e ordem) de cartas no tableau e estoque.

4.3. Tela de encerramento e Salvamento de Configuração

- Quando o jogador solucionar o jogo, ou quando o mesmo apertar a tecla ESC, o programa deverá ir para uma tela de encerramento.
- Nesta tela, o jogador poderá escolher **salvar** a configuração jogada, que corresponde ao embaralhamento inicial das cartas. O nome do arquivo a ser salvo deve ser informado pelo jogador.

4.4. Controle por parte do jogador

- O jogador moverá o cursor através das **teclas direcionais** ($\uparrow, \downarrow, \rightarrow, \leftarrow$). O cursor poderá passar por cada uma das pilhas de cartas na mesa (no tableau, na fundação, estoque ou descarte).
- O jogador selecionará uma jogada em uma pilha especifíca apertando a **tecla de espaço**.
- Se ao apertar espaço, o cursor estiver sobre o **estoque**, uma nova carta deve ser **repassada ao descarte**.
- Se ao apertar espaço, o cursor estiver sobre alguma outra pilha (que não o estoque), esta **pilha será selecionada**. Neste caso, o jogador deve mover o cursor até outra pilha e apertar espaço novamente para **selecionar a segunda pilha**.
  - Se for possivel fazer uma jogada movendo cartas da primeira pilha para o topo da segunda, então a jogada deve ser realizada.
  - Caso contrário, a seleção das pilhas deve ser cancelada.

4.5. Estruturas

- O programa deverá conter ao menos uma estrutura para representar **carta**, e essa estrutura conterá o **valor** e o **naipe** da carta.

4.6. Geração de configuração aleatória (embaralhamento)

- Quando o jogador escolher um Novo Jogo, uma nova configuração aleatória deve ser gerada, isto é, um embaralhamento das cartas deve ser realizado.
- Para tal é preciso embaralhar um **vetor** com as 52 cartas existentes, representando o Baralho.
- Note que o embaralhamento é uma **permutação**, ou seja, não pode haver repetição das cartas. Todas as cartas originais devem continuar existindo no vetor (mas em uma nova posição gerada aleatoriamente)

4.7. Vetores e Matrizes

- As pilhas de cartas deverão ser implementadas usando vetores e matrizes, declarados na *main*.
  - Evitar uso de variáveis globais.
- Idealmente, a pilha de cartas do **estoque** e do **descarte** deverão ser vetores de ponteiros da estrutura carta, enquanto que as pilhas da **fundação** e do **tableau** deverão ser matrizes de ponteiros da estrutura carta.
  - Os vetores e matrizes deverão ter tamanhos compatíveis com a quantidade de cartas que podem receber
  - Os tamanhos devem ser definidos via `define`.
- Cada elemento destes vetores e matrizes pode apontar para **uma carta** do baralho ou para **nenhuma** carta.
  - Deve-se garantir que todos os elementos que não apontam para nenhuma carta tenham valor NULL.
- A cada instante do jogo, uma carta do baralho só pode ser **apontada por exatamente um elemento** (de algum vetor ou matriz qualquer), ou seja, é proibida a repetição de cartas no jogo.
- A movimentação de uma carta de uma pilha para outra deverá ser feita através da cópia do ponteiro para esta carta.

4.8. Modularização

- Devido ao grande tamanho esperado, é muito importante que o programa seja altamente modularizado através de **funções**
  - As variáveis necessárias (vetores, matrizes, etc) devem ser cuidadosamente passadas como parâmetro na chamada das funções.
- Para uma melhor modularização do código, sugere-se também a organização do programa em **módulos descritos em diferentes arquivos** (por exemplo, separando o gerenciamento do jogo, desenho na tela, etc)
  - Cada módulo deve ter interfaces bem definidas (arquivos `.c` e `.h` para cada módulo).
- Abaixo estão algumas funções que podem ser desenvolvidas neste trabalho:
  - função para desenhar uma carta em uma dada posição da tela;
  - função para desenhar a mesa com todas as cartas, chamando outras funções (ex: de desenhar carta);
  - funções para desenhar tela inicial e tela final;
  - funções para salvar e carregar configuração de jogo;
  - função para movimentar cursor a partir da tecla lida;
  - função para gerar o embaralhamento das cartas;
  - função para inicializar as pilhas de cartas;
  - função para movimentar uma carta (deve receber quais as duas pilhas envolvidas na operação)e somente fazer a movimentação se for válida;
  - função para checar se jogo foi concluído;
  - etc.

5. Tarefas extras

Segue abaixo algumas sugestões de tarefas extras que podem ser implementadas. Embora opcionais, essas tarefas podem melhorar a avaliação final do seu trabalho. Também fique a vontade para implementar outras funcionalidades que achar interessante (mas não esqueça dos requisitos mínimos).

5.1. Armazenamento em arquivo dos scores dos jogos

- Sempre que um jogo for encerrado (quer seja porque o jogador conclui com sucesso, ou porque ele desistiu do jogo), o score do jogo deve ser **salvo em um arquivo** contendo todos os scores já jogados.
- Permitir que o jogador veja as 10 maiores pontuações já obtidas (lendo os valores do arquivo de scores). Mostre **em ordem decrescente** na tela essas 10 pontuações e os nomes dos jogadores que as conseguiram.

5.2. Carregamento e salvamento do estado de um jogo

- Permitir que o jogador interrompa um jogo no meio, guarde o estado do jogo em um arquivo (binário) e posteriormente continue o jogo (em outra execução do programa) lendo as informações salvas.
  - informações a serem salvas: distribuição das cartas em cada uma das pilhas, score, posição do cursor

5.3. Incluir variações do jogo de Paciência

- Virar 3 cartas do estoque por vez sobre o descarte o que exige retirar a primeira carta para acessar a segunda, e retirar a segunda para acessar a terceira
- Estabelecer um número limitado de "recicladas" do estoque a partir do descarte.

5.4. Aprimoramento da interface

- Fazer com que o controle do jogo seja feita através do MOUSE. Nesse caso, a implementação do controle via teclado (definido como requisito mínimo do trabalho) pode ser reduzida, mantendo obrigatoriamente apenas as funcionalidades que não podem ser feitas via mouse.
- Uso de alguma biblioteca para fazer a visualização do jogo com imagens, ao invés de somente modo texto. Por exemplo, pode se usar a biblioteca multi-plataforma Allegro (https://liballeg.org/) para criação de interfaces gráficas.