**MAC 5710 - Estruturas de Dados**
Primeiro semestre de 2006

Primeiro Exercício-Programa – Devolução: 31 de março

## FreeCell

Nesse exercício-programa vocês deverão implementar um jogo chamado *FreeCell* que descreveremos a seguir.

Usa-se no jogo um baralho (52 cartas). O objetivo é mover todas as cartas das pilhas de jogo para as pilhas de saída usando quatro posições auxiliares (*free cells*) para guardar temporariamente uma carta.

Existem quatro pilhas de saída, uma para cada naipe. As cartas nas pilhas de saída devem estar em ordem (ás, dois, três, etc).

A área de jogo do *FreeCell* consiste de quatro pilhas de saída, quatro posições auxiliares e o baralho, que é distribuído em oito pilhas de jogo no início do jogo, com todas as cartas abertas.

As cartas podem ser movidas para três lugares:

- para uma posição auxiliar desocupada: qualquer carta do topo de uma pilha de jogo.
- para uma das pilhas de saída: qualquer carta de uma posição auxiliar ou do topo de uma pilha de jogo. Movimentos para uma pilha de saída devem ser feitos em ordem do menor para o maior, mesmo naipe. Áses sempre podem ser movidos para uma pilha de saída vazia.
- para o topo de uma pilha de jogo: qualquer carta de uma das posições auxiliares ou do topo de uma outra pilha de jogo. Movimentos para uma pilha de jogo devem ser feitos em ordem do maior para o menor, alternando a cor do naipe.

O jogo termina ou quando todas as cartas foram movidas para as pilhas de saída ou quando não há movimento que permita mais alguma carta ser movida para uma das pilhas de saída.

Acredita-se, embora nunca tenha sido provado, que todo jogo pode ser vencido.

Faça um programa em `C` que permita ao usuário jogar *FreeCell* como descrito acima, com a possibilidade de jogar várias partidas.

**Todas as pilhas necessárias para o seu programa devem ser armazenadas em um único vetor com 52 posições**, e para o tratamento de *overflow* você deve usar um algoritmo de realocação de tabelas seqüênciais, descrito no livro do Knuth, vol. I, seção 2.2.2 (pags. 244 a 251). Observe que, devido ao tamanho máximo do vetor, ocorrerão muitas colisões entre as pilhas, portanto seu tratamento deve ser o mais eficiente possível.

Para vocês que não conhecem jogos, as cartas de um baralho estão divididas em quatro naipes: ouros, copas, paus e espadas. Ouros e copas são naipes vermelhos, enquanto que paus e espadas são naipes pretos. Para cada naipe, existem 13 cartas: A (ás - 1), 2, 3, 4, 5, 6, 7, 8, 9, 10, J (valete - 11), Q (dama - 12) e K (rei - 13). Escolha uma implementação o mais eficiente possível para armazenar as cartas.

Seu programa deve imprimir o conteúdo das pilhas a cada jogada, assim como uma mensagem dizendo que movimento o usuário realizou. As pilhas de jogo **deverão** ser impressas em forma de pilhas, uma ao lado da outra. Obviamente seu programa deverá ser bem documentado, e a forma de utilizá-lo também será avaliada. Faça consistência de dados, ou seja, o seu programa deve avisar quando o usuário tenta fazer um movimento de cartas inválido e permitir que ele tente uma nova jogada.

Quem quiser treinar um pouco de jogo antes de começar a programar ☺ poderá testar a versão para *Windows* ou uma versão Java feita há muitos anos, pelo *Caetano Jimenez Carezzato*, um aluno muito bom da nossa graduação. As duas versões estão disponíveis na página da disciplina.