Enunciado

Laboratórios de Informática II

Laboratórios de Programação II

1 Requisitos

Requisitos para todas as entregas

Requisitos principais

- Funcionalidade;
- Qualidade de software;
- Reutilização de código.

Requisitos para todas as entregas

Projeto

- Deve ser desenvolvido em C;
- Deve executar na máquina virtual disponibilizada no endereço `lambda.di.uminho.pt`;
- Todas as funções **não podem ultrapassar** a **complexidade ciclomática** de 10;
- Todas as funções **não podem ultrapassar** as 15 instruções;
- **Não podem existir variáveis globais**;
- **Não podem usar goto, break ou continue**;
- O código deve ser **modular**;
- Deve existir uma separação entre a **lógica** e o **interface**;
- Todas as funções da lógica **têm que ser** testadas com **`cunit`**;
- Todas as funções **têm que ser** documentadas.

Momentos de Avaliação

- Há quatro momentos de avaliação;
- Cada grupo deve criar um repositório git local na máquina virtual **lambda.di.uminho.pt**;
- O código deve ser submetido no repositório git da máquina até ao **Prazo**;
- O Momento de Avaliação correspondente a essa etapa será na semana que se encontra na coluna **Avaliação**;
- Durante o momento de avaliação, o docente irá buscar o commit mais recente anterior ao **Prazo** para proceder à avaliação.

Momentos de Avaliação

Etapa	Percentagem	Prazo	Avaliação
1	25%	22 de Março	23 a 28 de Março
2	25%	19 de Abril	20 a 25 de Abril
3	25%	24 de Maio	25 a 30 de Maio
Final	25%	31 de Maio	1 a 5 de Junho

Avaliação

Penalização

- Se um projeto não cumprir qualquer dos requisitos, este não será avaliado
- Se nenhum aluno for capaz de responder a uma pergunta sobre o código, a avaliação do projeto será **NULA**
- Os docentes reservam-se o direito de chamar os alunos para uma nova defesa oral, individualmente ou em grupo, caso surja alguma dúvida sobre o projeto
- Caso os alunos se recusem a responder a qualquer pergunta, ou se recusem a comparecer, serão reprovados à UC

Avaliação Individual

- O não comparecimento às defesas das etapas implica a *não avaliação* (**ZERO**) nessa etapa;
- O não ser capaz de responder a qualquer pergunta numa etapa implica o mesmo que a alínea anterior;
- A avaliação de cada elemento deverá ser uma **percentagem** da avaliação do projeto.

Avaliação de pares (DESCONTINUADO!!!)

Esta avaliação foi descontinuada devido a problemas com a ferramenta

- ~~Cada elemento deverá avaliar os seus colegas em cada etapa;~~
- ~~Caso um elemento não avalie os seus colegas, ele terá **ZERO** nessa etapa.~~

2 Etapas

Golf

Jogo

- Exemplo
- 7 pilhas de 5 cartas são dispostas;
- A pilha de descarte começa com uma carta visível;
- Pode-se pegar numa carta do topo da pilha e colocar na pilha de descarte desde que seja o valor imediatamente acima ou abaixo do que está no topo da pilha de descarte;
- A qualquer altura pode-se biscar uma carta do baralho e colocar no topo da pilha de descarte.

Implementação

- Command Line Interface (CLI)
- Crie os módulos e estruturas de dados que servirão de base para o projeto

Simple Simon

Jogo

- Exemplo
- Layout de colunas;
- O foco é a movimentação de sequências de cartas entre colunas e a gestão de espaços vazios;

Implementação

- Deverá reutilizar o **máximo** de lógica de manipulação de cartas desenvolvida na etapa anterior.

Terceira etapa

Objectivo

- O seu programa deverá ler todas as Paciências de uma pasta.
- A pasta terá o nome **paciencias**.
- Cada Paciência será descrita num ficheiro de texto ASCII.
- O seu programa deverá permitir ao utilizador escolher uma das Paciências para jogar.
- Deve permitir voltar atrás.
- Deve permitir fazer *save* e *load* de estados de jogo num formato específico.

DSL para descrever Paciências

- Exclusivamente uma paciência por ficheiro.
- O ficheiro consiste num conjunto de comandos separados por linhas (`\n`).
- Um **comando** consiste num nome (do comando) seguido de **parâmetros** separados por espaços.

Terceira etapa (ii)

- Os parâmetros são de três tipos: identificadores (`[a-zA-Z]+`), numéricos (`[0-9]+`) ou *flags* (`[a-zA-Z+<>^_*~\[\]=1]+`).
- Cada símbolo numa *flag* denota uma propriedade e a *flag* denota a conjunção de todas as propriedades (símbolos) que a compõem.
- Linhas em branco são ignoradas.
- Qualquer texto após `#` deve ser ignorado - servem de **comentários**.

Exemplo completo (Golf)

```
JOGO Golf             # Esta Paciência chama-se Golf
BARALHOS 1            # Joga-se com 1 baralho
TIPO TAB =            # Pilhas no tabuleiro têm as cartas todas visíveis
TIPO STOCK _          # Pilha de stock não tem cartas visíveis
TIPO DESCARTE =       # Pilha de descarte mostra todas as cartas
INIT TAB 5            # Há 7 pilhas de tabuleiro, cada uma com 5 cartas inicialmente
INIT TAB 5
INIT TAB 5
INIT TAB 5
INIT TAB 5
INIT TAB 5
INIT TAB 5
INIT DESCARTE 1       # Pilha de descarte começa com 1 carta
INIT STOCK 16         # Pilha de stock começa com 16 cartas
MOV STOCK DESCARTE * # Podemos sempre mover uma carta do stock para o descarte
MOV TAB DESCARTE ~    # Podemos mover do tabuleiro para o descarte se o valor for -1/+1
WIN TAB 0             # Vitória quando todas as pilhas do tabuleiro estiverem vazias
```

Comandos da DSL

Comando `JOGO <nome>`

- Define o nome da Paciência especificada no ficheiro actual.
- Exemplo:
  - `JOGO SimpleSimon`

Comando `BARALHOS <n>`

- Indica o número de baralhos requeridos pela Paciência (`n >= 1`).
- Exemplo:
  - `BARALHOS 2`

Comandos da DSL (ii)

Comando `TIPO <tipo-pilha> <flags>`

- Define um tipo de pilha chamado `<tipo-pilha>` com as propriedades descritas nas `<flags>`.
- *Flags*:
  - `=` Todas as cartas da pilha são visíveis.
  - `_` Nenhuma carta da pilha é visível.
  - `^` Apenas a carta do topo é visível.
  - `1` A pilha pode conter, **no máximo**, uma carta. De outra forma, não há restrições quanto ao número de cartas que uma pilha pode ter.
- Exemplo:
  - `TIPO TAB 1=  # pilhas TAB podem conter 0 ou 1 cartas. As cartas são visíveis`
  - `TIPO STOCK _ # pilhas STOCK podem conter 0 ou mais cartas - porém, invisíveis`

Comandos da DSL (iii)

Comando `INIT <tipo-pilha> <n>`

- Indica que deve existir uma pilha do tipo (*cf.* `TIPO`) `<tipo-pilha>` com n (`n >= 0`) cartas inicialmente.
- Este comando pode ser utilizado múltiplas vezes ao longo do ficheiro. De cada vez que este comando é utilizado, uma nova instância daquele tipo de pilha deverá ser criada com o número de cartas indicado.
- Exemplo:

```
INIT TAB 5 # há uma pilha do tipo TAB com 5 cartas
INIT TAB 2 # há outra pilha, também do tipo TAB, com 2 cartas
```

Comandos da DSL (iv)

Comando `MOV <tipo-origem> <tipo-destino> <flags>`

- Descreve, através das `<flags>`, em que condições é possível mover cartas de uma pilha do tipo `<tipo-origem>` para uma pilha do tipo `<tipo-destino>`.
- Mover significa: retirar cartas contíguas do topo da pilha de origem para o topo da pilha de destino.
- A ausência de comandos `MOV` implica que a Paciência não admite jogadas (movimentos).
- Este comando pode ser utilizado múltiplas vezes ao longo do ficheiro. Quando uma Paciência admitir vários tipos de movimentos entre os mesmos tipos de pilhas, a regra aplicável corresponde à **disjunção** dos vários comandos `MOV`.
- As comparações de valor (`<`, `>`, `~`, `[`, `]`) são **lineares**, *i.e.* Ás e Rei não são adjacentes.
- *Flags*:
  - `*` Não há restrições, *i.e.* é sempre possível mover cartas entre os dois tipos de pilhas.

Comandos da DSL (v)

- `+` Pode ser movida uma sequência de cartas. Sem esta *flag*, só é possível mover uma carta.
- `[` As cartas a mover devem estar **ordenadas de forma decrescente**, por valores consecutivos.
- `]` As cartas a mover devem estar **ordenadas de forma crescente**, por valores consecutivos.
- `<` A carta de topo na sequência (eventualmente singular) a mover deve ter um **valor imediatamente inferior** à carta de topo no destino.
- `>` A carta de topo na sequência (eventualmente singular) a mover deve ter um **valor imediatamente superior** à carta de topo no destino.
- `~` Atalho para "`<` **ou** `>`"", *i.e.* o valor deve ser imediatamente superior ou imediatamente inferior.
- `m` As cartas a mover **devem pertencer ao mesmo naipe**.

Comandos da DSL (vi)

- `M` A carta de topo na sequência (eventualmente singular) a mover **deve pertencer ao mesmo naipe** da carta no topo do destino.
- `x` As cartas a mover **devem ser de naipes alternados**, *i.e.* cada par de cartas consecutivas são de naipes diferentes.
- `X` A carta de topo na sequência (eventualmente singular) a mover **não deve pertencer ao mesmo naipe** da carta no topo do destino.
- `c` As cartas a mover **devem ter a mesma cor**.
- `C` A carta de topo na sequência (eventualmente singular) a mover **deve ter a mesma cor** da carta no topo do destino.
- `d` As cartas a mover **devem ter cores alternadas**, *i.e.* cada par de cartas consecutivas têm cores diferentes.

Comandos da DSL (vii)

- `D` A carta de topo na sequência (eventualmente singular) a mover **não deve ter a mesma cor** da carta no topo do destino.
- `V` A pilha de destino **deve estar vazia**.
- `a` A carta de **topo** na sequência (eventualmente singular) a mover **deve ser um Ás**.
- `A` A carta de **fundo** na sequência (eventualmente singular) a mover **deve ser um Ás**.
- `k` A carta de **topo** na sequência (eventualmente singular) a mover **deve ser um Rei**.
- `K` A carta de **fundo** na sequência (eventualmente singular) a mover **deve ser um Rei**.

- Exemplo:
  - `MOV A B <X    # valor imediatamente inferior e naipe alternado`
  - `MOV X Y aV    # apenas move Ás se Y estiver vazia`
  - `MOV A B +[m<  # ordenada decr, mesmo naipe e valor inferior ao destino`

Comandos da DSL (viii)

Comando `AUTO <tipo-origem> <tipo-destino> <flags>`

- Semelhante ao comando `MOV`, mas denota uma movimentação **automática** que ocorre sempre que as condições descritas pelas `<flags>` se verificarem.
- Estas movimentações devem ser testadas (em cadeia) após qualquer jogada do utilizador, sendo realizadas automaticamente, sem contarem como uma jogada.
- Exemplo:
  - `AUTO TAB FUND +[mKaV # move uma sequência K...A (Ás no topo) do mesmo naipe`

Comando `WIN <tipo-pilha> <n>`

- Descreve, através do número de cartas de um tipo de pilha, um critério de vitória.

Comandos da DSL (ix)

- Este comando pode ser utilizado múltiplas vezes ao longo do ficheiro, sendo a regra aplicável a **conjunção** dos vários comandos `WIN`.
- Exemplo:
  - `WIN TAB 0 # termina em vitória quando todas as pilhas TAB estiverem vazias`
  -
    ```
    # termina em vitória quando as pilhas FOO só tiverem 1 carta
    # e as pilhas BAR estiverem vazias
    WIN FOO 1
    WIN BAR 0
    ```

Formato dos jogos gravados

- Consiste num ficheiro de texto ASCII separado por linhas (`\n`).
- A primeira linha consiste no nome do ficheiro da Paciência encontrado na pasta **paciencias**.
- De seguida, haverá tantas linhas quantos comandos `INIT` no ficheiro da Paciência correspondente:
  - A primeira linha indica que cartas a primeira pilha feita `INIT` contém.
  - Restantes linhas, *idem*.
- Em cada linha, as cartas são separadas por espaços.
  - Cada carta é representada no formato `<valor><naipe>` (sem espaços):
    - `<valor>` é um de `A`, `2`, `3`, ..., `10`, `J`, `Q`, `K`.
    - `<naipe>` é um de `C` (Paus), `D` (Ouros), `H` (Copas) e `S` (Espadas).
  - A ordem das cartas em cada linha é do **fundo** para o **topo** da pilha, *i.e.* a carta mais à direita é a carta no topo.
  - Se a n-ésima pilha estiver vazia, a linha $n + 1$ correspondente é também vazia.
- Ao contrário do ficheiro das Paciências, este ficheiro **não admite comentários**. Os comentários usados no exemplo que se segue são apenas ilustrativos.

Formato dos jogos gravados (ii)

Exemplo

```
golf.paciencia  # nome do ficheiro que descreve a paciência
2C 7D 3H AS     # 4 cartas actualmente na primeira pilha; Ás de espadas no topo
                # segunda pilha está vazia
10S             # 3ª pilha só tem uma carta
```

3 Ferramentas

pmccabe

Vamos utilizar:

- *Modified McCabe Cyclomatic Complexity*
- *Statements in function*

```
rcm@lambda:~/guioes/G1$ pmccabe -v *.c
Modified McCabe Cyclomatic Complexity
|   Traditional McCabe Cyclomatic Complexity
|       |    # Statements in function
|       |        |   First line of function
|       |        |       |   # lines in function
|       |        |       |       |  filename(definition line number):function
|       |        |       |       |           |
1 1 3 5 7 main.c(5): main
1 1 4 4 7 quad.c(4): raizes
```

4 Exemplos completos da DSL

Exemplo completo (Simple Simon)

```
JOGO SimpleSimon
BARALHOS 1
TIPO TAB =
TIPO FUND =
INIT TAB 8
INIT TAB 8
INIT TAB 8
INIT TAB 7
INIT TAB 6
INIT TAB 5
INIT TAB 4
INIT TAB 3
INIT TAB 2
INIT TAB 1
INIT FUND 0
INIT FUND 0
INIT FUND 0
```

Exemplo completo (Simple Simon) (ii)

```
INIT FUND 0

# 1. uma carta pode ser movida se o seu valor for imediatamente inferior
# 2. o mesmo se aplica a sequências do mesmo naipe, ordenadas em ordem decrescente
# 3. qualquer carta pode ser movida para uma pilha vazia
# 4. o mesmo se aplica a uma sequência ordenada decrescentemente do mesmo naipe
MOV TAB TAB <
MOV TAB TAB +[m<
MOV TAB TAB V
MOV TAB TAB +[mV

# sequências completas K...A do mesmo naipe vão automaticamente para uma fundação vazia
AUTO TAB FUND +[mKaV

WIN TAB 0 # vitória quando as pilhas do tabuleiro ficarem vazias
```

Exemplo completo (FreeCell)

```
JOGO FreeCell
BARALHOS 1
TIPO TAB =
TIPO CELL 1=
TIPO FUND =

INIT TAB 7
INIT TAB 7
INIT TAB 7
INIT TAB 7

INIT TAB 6
INIT TAB 6
INIT TAB 6
INIT TAB 6

INIT CELL 0
```

Exemplo completo (FreeCell) (ii)

```
INIT CELL 0
INIT CELL 0
INIT CELL 0

INIT FUND 0
INIT FUND 0
INIT FUND 0
INIT FUND 0

MOV TAB TAB D<
MOV TAB TAB V
MOV TAB FUND aV
MOV TAB FUND M>
MOV TAB CELL V

MOV CELL TAB D<
MOV CELL TAB V
```

Exemplo completo (FreeCell) (iii)

```
MOV CELL FUND aV
MOV CELL FUND M>

WIN TAB 0
WIN CELL 0
```

Exemplo completo (Klondike simplificado)

```
JOGO KlondikeSimplificado
BARALHOS 1
TIPO TAB =
TIPO FUND ^
TIPO STOCK _
TIPO DESCARTE ^

INIT TAB 7
INIT TAB 6
INIT TAB 5
INIT TAB 4
INIT TAB 3
INIT TAB 2
INIT TAB 1

INIT FUND 0
INIT FUND 0
```

Exemplo completo (Klondike simplificado) (ii)

```
INIT FUND 0
INIT FUND 0

INIT DESCARTE 0
INIT STOCK 24

MOV STOCK DESCARTE *

MOV TAB TAB d<
MOV TAB TAB +[d<

MOV TAB TAB kV
MOV TAB TAB +[dVK

MOV TAB FUND aV
MOV TAB FUND M>
```

Exemplo completo (Klondike simplificado) (iii)

```
MOV DESCARTE FUND aV
MOV DESCARTE FUND M>

MOV DESCARTE TAB d<
MOV DESCARTE TAB kV

MOV FUND TAB d<
MOV FUND TAB kV

WIN FUND 13
```

5 Sugestões implementação Etapa 3

Parsing

- Parsing ficheiro da paciência
  - utilizar `fgets` para ler linha
  - substituir o primeiro `#` por zero para remover os comentários
  - testar através do `sscanf` o tipo de comando e invocar função correspondente

Estado do jogo

- representado numa estrutura dinâmica, respeitando a ordem definida na DSL (comandos INIT)
- Movimentos válidos
  - no parsing da paciência, inicializar estrutura para listar os movimentos válidos (comandos MOV)
  - cada elemento da lista específica: tipo de origem; tipo de destino; e um conjunto de flags que caracterizam as restrições associadas
  - a cada jogada, percorre-se a estrutura à procura do primeiro elemento que corresponda aos tipos de pilhas envolvidos e valide todas as restrições