E-fólio A | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Introdução à Programação

CÓDIGO: 21173

DOCENTE: José Coelho

A preencher pelo estudante

NOME: Luís Guilherme Sousa de Oliveira

N.º DE ESTUDANTE: 2000274

CURSO: Unidade Curriculares Isoladas – Engenharia Informática

TRABALHO / RESOLUÇÃO:

Nota prévia

O código e informações apresentados neste relatório dizem respeito, maioritariamente, ao trabalho desenvolvido até à alínea 4 do e-folio A excluindo a parte do jogador automático (JA). Apesar de se ter começado a desenvolver as regras para o JA não foi possível finalizá-la. Dessa forma todos os testes apresentados excluem o JA.

Fluxo de programa e função Main

O fluxo implementado no e-fólio é o seguinte:

1. Criação de um tabuleiro vazio;
2. Implementação das 14 peças no tabuleiro com a devida validação de entrada;
3. Movimentação das peças, com a devida validação das casas de entrada e dos movimentos pretendidos;
4. Impressão do tabuleiro final.

Dessa forma a função *main* funciona da seguinte forma:

- Obtém uma string utilizando a função *gets;*

- Cria um vetor *tabuleiro* através da função *criar_tabuleiro*;

- Verifica a validade das duas primeiras casas a entrar (reis) e devolve “Posição Inválida” caso um dos reis seja posicionado numa casa inválida. No anexo II encontram-se dois testes que mostram esta validação ao provar que sem rei não há jogo mas sem peões o jogo começa.

- Se estiver tudo em condições, são inseridas as peças (reis e peões) no tabuleiro com o procedimento *inserir_pecas*;

- Seguidamente movimentam-se as peças utilizando *mover_pecas*, que devolve o número final de jogadas;

- Como as peças brancas são sempre as primeiras a jogar, consegue-se saber através do número de jogadas saber quem foi a equipa vencedora (brancas tem jogadas em número ímpar). No anexo II verificam-se diferentes possíveis jogadas e, por exemplo, caso a primeira jogada seja inválida ganha a equipa das pretas com 0 jogadas válidas.

- Por fim imprime-se o tabuleiro com a função *imprimir_output* juntamente com o texto referente ao vencedor e ao número de jogadas.

Na função *main* assumiu-se uma entrada de uma string única (utilizando a função *gets*) com as casas do tabuleiro separadas por espaços. Utilizou-se como

tamanho deste vetor o valor 200 de dimensão pois esta recebe, no mínimo, uma entrada de 42 carateres (14 peças de dois carateres e os espaços entre elas). Como cada movimento de peça utiliza cinco carateres (duas casas – de e para – e o espaço) então 15 movimentos daria um total necessário de 117 carateres. Num dos testes apresentados no anexo II o jogo apresenta 19 jogadas válidas, ou seja, 20 duplas de casas de forma a mostrar a disponibilidade da dimensão vetor.

Foi decidido utilizar um apontador para o **tabuleiro* para poder ser criado com a *criar_tabuleiro* e alterado posteriormente.

Descrição das funções

De seguida é descrito o que cada função/procedimento recebe de argumento(s), o que esta obtém, como é que obtido este resultado e alguns pormenores da função (caso existam). O código de todas as funções (inclusive a *main*) está no anexo I.

criar_tabuleiro:

Função sem argumentos e que produz um vetor com 30 entradas correspondentes a casas vazias, ou seja, pontos ('.'). Foi decidido isolar esta função por uma questão de leitura do código. Foi necessário tornar o vetor de saída como *static* para poder ser utilizado como saída.

verificar_posicao:

Função criada para a alínea A recebe como argumento um vetor carater e procede à sua validação. Foi decidido validar o vetor sobre três componentes a) dimensão da string pois esta tem de ser inferior a 2, b) verificar se o primeiro carater da string estava entre a letra 'a' e 'f' e c) verificar se o segundo carater estava entre o numero 1 e 5. A função devolve o número 0 caso o vetor não seja válido e 1 caso seja válido.

separar_casas

Esta função recebe uma string (vetor carater), separa esse vetor utilizando o espaço como separador e devolve um vetor string. Para obter este comportamento foi necessário utilizar a função *strtok* da biblioteca *string.h* e fazer chamadas sucessivas (com uma função *While*) dado que, tal como é indicado em "Coding Games and Programming Challenges to Code Better" [1], "*strtok needs to be called several times to split a string".*

[1] https://www.codingame.com/playgrounds/14213/how-to-play-with-strings-in-c/string-split

Tal como em *criar_tabuleiro* atribuiu-se ao vetor string uma característica *static*.

Criou-se esta função pois assumiu-se uma entrada inicial única no programa com as casas separadas por espaços (já referido anteriormente).

converte_localizacao_em_numero

Esta função recebe um vetor carater e devolve um número de 0 a 29. Esta função procede à validação do vetor de entrada utilizando a *verificar_posicao* e, se for válido, procede a um ciclo e à análise de duas condições para transformar as strings de a1 a f5 em números de 0 a 29. Esta função surge na necessidade de chamar esta "transformação de casas em números" várias vezes ao longo do programa.

inserir_pecas

Este procedimento recebe dois vetores carater como entrada (*casas[]* e *tabuleiro[]*) e procede à transformação do vetor tabuleiro "inserindo" os reis e peões. A função *separar_casas* é chamada dentro deste procedimento e as 15 primeiras casas (ou seja, 44 primeiros carateres – dois carateres por casa e um espaço entre eles) são usadas para as peças. Depois procede-se a dois ciclos – de 0 a 14 para as 15 peças e de 0 a 29 para as 30 casas do tabuleiro – substituindo os '.' pelas peças reis e peões.

validar_movimento_peca

Esta função recebe duas strings (de e para onde se deslocam as peças) e o vetor carater e devolve 0 caso o movimento não seja válido, 1 caso seja um movimento válido normal e -1 caso seja tomado o rei.

O objetivo principal desta função é servir de validação ao movimento das peças. Esta função chama, inicialmente, a função *converte_localizacao_em_numero* para poder transformar os movimentos em inteiros.

Como referido no enunciado, o movimento dos peões é uma casa na vertical (direcção única conforme a equipa) para uma casa vazia e, caso não se queira "tomar um adversário", é uma casa na horizontal. Os reis podem andar para qualquer casa adjacente, mas apenas uma casa. Foi com este pensamento que se valida o movimento utilizando os condicionais *switch* e *if*.

Visto que as direções e sentidos são muito utilizadas ao longo da função decidi criar os vetores vertical e diagonal (como uma casa na horizontal é só um número menos/mais outro e só é usada para os reis decidi não ocupar espaço em memória com esta informação).

Foi decidido separar a tomada do rei de outro movimento pois a tomada do rei tem de ser considerada uma jogada válida, mas também o fim do jogo.

regras_JA

Este procedimento foi criado especificamente para a parte de Jogador Automático (JA) mas não foi, infelizmente, concluído.

O pensamento seguido para fazer esta parte foi o seguinte:

- O procedimento recebia o inteiro *jogadas* que permitia saber por que equipa o JA estava a jogar;

- Num ciclo de 0 a 29 obtêm-se as localizações de todos reis e peões;

- Procediam-se a vários ciclos para verificar os vários níveis das regras do JA e utilizava-se essa informação para ser utilizada numa matriz;

- Procedia-se a um ciclo final que fazia a troca de símbolos de casas conforme o disponibilizado na matriz anteriormente referida.

mover_pecas

Esta função recebe dois vetores carater como entrada (*casas[]* e *tabuleiro[]*) e devolve um inteiro referente ao número de jogadas válidas. Este número de jogadas é usado na função *main* para decidir quem foi a equipa vencedora e, dessa forma, apresentar o texto final de vitória.

Procede a um ciclo começando após as 14 peças e valida se as jogadas (casa de origem e de destino) são de um Jogador Automático (JA) e se é uma jogada válida utilizando a função *validar_movimento_peca.*

Se for JA segue para *regras_JA*, se for uma jogada inválida (ou seja, é devolvido 0 da função *validar_movimento_peca)* a função é interrompida e se for uma jogada válida procede-se à troca de símbolos e incrementa-se o inteiro static *jogadas* em 1 valor. Se, para além da jogada ser válida, a função *validar_movimento_peca* devolver o valor -1 então o jogo é parado pois o rei foi tomado.

De realçar o iterador i que, apesar de estar dentro do ciclo precisa de ser incrementado uma segunda vez se a jogada for válida pois a origem é casa *i* mas o destino é casa *i+1*;

imprimir_output

Procedimento que recebe como entrada um vetor carater e imprimi o tabuleiro utilizando dois ciclos (de 1 a 6 para as linhas e de 0 a 6 para as colunas) utilizando o carater especial *n* para as linhas. Neste procedimento utilizou-se a variável k (que vai de 0 a 29) dentro dos ciclos para poder a) obter a lateral esquerda através do cálculo 5-k/6 e b) obter o carater dentro do tabuleiro. No fim imprime-se a lateral inferior do tabuleiro.

ANEXOS

Anexo I – Código da alínea 4

```
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include <limits.h>
#include <stdbool.h>

//Função que cria um vetor caracter com 37 casas
char* criar_tabuleiro() {
    static char posicoes[31];
    int i;
    for (i=0;i<30;i++)
        posicoes[i]='.';
return posicoes;
}
//Função que verifica se a casa está no tabuleiro
int verificar_posicao (char casa[]) {
    //Avaliar dimensao de input
    if(strlen(casa)>2)
        return 0;
    // Avaliar se a primeira char fica entre a e f
    else if (casa[0]<'a'||casa[0]>'f')
        return 0;
    // Avaliar se a segunda char fica entre 1 e 5
    else if (casa[1]<'1'||casa[1]>'5')
        return 0;
    else
        return 1;
}
//Função que recebe um string com espacos, separa-
o e devolve um vector string
char** separar_casas(char casas[]) {
    static char *localizacao[200];
    char *casa_sem_espacos = strtok(casas, " ");
    int c;

    c=0;
    while (casa_sem_espacos != NULL)
        {
        localizacao[c]=casa_sem_espacos;
        c++;
        casa_sem_espacos = strtok(NULL, " ");
```

```
        }
    return localizacao;
}
// Função que recebe uma casa do tipo letra_numero e devolve um int
eiro
int converte_localizacao_em_numero(char casa[]) {
    int i;
    char *numero[7]={"0","1","2","3","4","5"};
    char *letra[7]={"a","b","c","d","e","f"};

    // Apenas transforma letras em inteiros se for válido
    if (verificar_posicao(casa)==1)
        for (i=0;i<35;i++)
            if((casa[0]==*letra[i%6])&&(casa[1]==*numero[5-i/6]))
                return i;
    return -1;
}
// Procedimento que coloca 15 peças no vetor tabuleiro
void inserir_pecas(char casas[],char tabuleiro[]) {
    int i,c;
    char *pecas[15]={"R","r","P","P","P","P","P","P","p","p","p","p
","p","p"};
    char **localizacao=separar_casas(casas);

    // Inserir as 15 peças ...
    for (c=0;c<14;c++)
        // ... no tabuleiro com 30 casas
        for (i=0;i<30;i++)
            if (converte_localizacao_em_numero(localizacao[c])==i)
                {
                tabuleiro[i]=*pecas[c];
                }
}
//Funcao que verifica se o movimento da peca é possível
int validar_movimento_peca(char *origem, char *destino, char tabule
iro[]){
    int de_int=converte_localizacao_em_numero(origem);
    int para_int=converte_localizacao_em_numero(destino);
    int vertical[2];
    int diagonal[4];

    // Descer e subir
    vertical[0]=(para_int-de_int)%6;
    vertical[1]=(de_int-para_int)%6;
    // Baixo_esquerdo , baixo direito
    diagonal[0]=((para_int-1)-de_int)%6;
```

```
    diagonal[1]=((para_int+1)-de_int)%6;
    // Cima_direito , cima_esquerdo
    diagonal[2]=((de_int-1)-para_int)%6;
    diagonal[3]=((de_int+1)-para_int)%6;

    if (verificar_posicao(origem)==0||verificar_posicao(destino)==0
)
        return 0;
    else
        switch(tabuleiro[de_int])
        {
        case '.':
            return 0;
            break;
        case 'p':
            if (tabuleiro[para_int]=='R'&&(diagonal[0]==0||diagonal
[1]==0)&&(de_int-para_int<8&&para_int-de_int<8))
                return -1;
            else if (tabuleiro[para_int]=='P'&&(diagonal[0]==0||dia
gonal[1]==0)&&(de_int-para_int<8&&para_int-de_int<8))
                return 1;
            else if (tabuleiro[de_int]=='.'&&vertical[0]==0&&(de_in
t-para_int<8&&para_int-de_int<8))
                return 1;
            else
                return 0;
            break;
        case 'P':
            if (tabuleiro[para_int]=='r'&&(diagonal[2]==0||diagonal
[3]==0)&&(de_int-para_int<8&&para_int-de_int<8))
                return -1;
            else if (tabuleiro[para_int]=='p'&&(diagonal[2]==0||dia
gonal[3]==0)&&(de_int-para_int<8&&para_int-de_int<8))
                return 1;
            else if (tabuleiro[para_int]=='.'&&vertical[1]==0&&(de_
int-para_int<8&&para_int-de_int<8))
                return 1;
            else
                return 0;
            break;
        case 'r':
            if (tabuleiro[para_int]=='R'&&(de_int-
para_int==1||para_int-
de_int==1||vertical[0]==0||vertical[1]==0||diagonal[0]==0|| diagona
l[1]==0||diagonal[2]==0||diagonal[3]==0)&&(de_int-
para_int<8&&para_int-de_int<8))
```

```
                return -1;
            else if (tabuleiro[para_int]=='p')
                return 0;
            else if (de_int-para_int==1||para_int-
de_int==1||vertical[0]==0||vertical[1]==0||diagonal[0]==0|| diagona
l[1]==0||diagonal[2]==0||diagonal[3]==0)
                return 1;
            else
                return 0;
            break;
        case 'R':
            if (tabuleiro[para_int]=='r'&&(de_int-
para_int==1||para_int-
de_int==1||vertical[0]==0||vertical[1]==0||diagonal[0]==0|| diagona
l[1]==0||diagonal[2]==0||diagonal[3]==0)&&(de_int-
para_int<8&&para_int-de_int<8))
                return -1;
            else if (tabuleiro[para_int]=='P')
                return 0;
            else if ((de_int-para_int==1||para_int-
de_int==1||vertical[0]==0||vertical[1]==0||diagonal[0]==0|| diagona
l[1]==0||diagonal[2]==0||diagonal[3]==0)&&(de_int-
para_int<8&&para_int-de_int<8))
                return 1;
            else
                return 0;
            break;
    }
return 1;
}
// Procedimento para o Jogador Automático
void regras_JA(char tabuleiro[], int jogadas)
{
    int rei[2];
    int peao[8][2];
    int i,j=0,k=0;
    int jogador,adversario;
    int movimento[4][5];

    if(jogadas%2==0)
      {
        jogador=1;
        adversario=0;
      }
    else
      {
```

```
        jogador=0;
        adversario=1;
      }

    // As casas pretas estão no [0] e o branco no [1]
    for (i=0;i<35;i++)
        switch (tabuleiro[i])
        {
        case ('r'):
            rei[0]=i;
            break;
        case ('R'):
            rei[1]=i;
            break;
        case ('p'):
            peao[j][0]=i;
            j++;
            break;
        case ('P'):
            peao[k][1]=i;
            k++;
            break;
        default:
            tabuleiro[i]='.';
            break;
        }

    //Tomar peao
    for (j=0;j<6;j++)
        for (k=0;k<6;k++)
            if (((peao[j][adversario]+1)-
peao[k][jogador])%6==0&&(peao[j][adversario]-peao[k][jogador])<7)
                {
                movimento[1][0]=peao[k][jogador];
                movimento[1][1]=peao[j][adversario];
                movimento[1][2]=1;
                printf("");
                break;
                }
            else if (((peao[j][adversario]-1)-
peao[k][jogador])%6==0&&(peao[j][adversario]-peao[k][jogador])<5)
                {
                movimento[1][0]=peao[k][jogador];
                movimento[1][1]=peao[j][adversario];
                movimento[1][2]=1;
                printf("");
```

```
                break;
                }


    for (j=0;j<4;j++)
        if (movimento[j][2]==1)
            {
            tabuleiro[movimento[j][0]]='.';
            tabuleiro[movimento[j][1]]='p';
            break;
            }
}

// Função que permite movimentar as peças no vetor tabuleiro
int mover_pecas(char casas[], char tabuleiro[]) {
    int i;
    static int jogadas;
    char peca;
    char *origem,*destino;

    for (i=14;separar_casas(casas)[i]!=0;i++)
        {
        if (separar_casas(casas)[i+1]!=0)
            {
            if (separar_casas(casas)[i][0]=='J'&&separar_casas(casa
s)[i][1]=='A')
                {
                regras_JA(tabuleiro,jogadas);
                jogadas++;
                }
            else if (validar_movimento_peca(separar_casas(casas)[i]
,separar_casas(casas)[i+1],tabuleiro)==0)
                break;
            else
                {
                origem=separar_casas(casas)[i];
                destino=(separar_casas(casas)[i+1]);

                peca=tabuleiro[converte_localizacao_em_numero(orige
m)];
                tabuleiro[converte_localizacao_em_numero(origem)]='
.';
                tabuleiro[converte_localizacao_em_numero(destino)]=
peca;
                i++;
                jogadas++;
```

```
                if (validar_movimento_peca(origem,destino,tabuleiro
)==-1)
                    break;
                }
            }
        else
            break;
        }
return jogadas;
}
//Procedimento para imprimir o tabuleiro final
void imprimir_output(char tabuleiro[]) {
    int i,j,k;

    k=0;
    for (i=1;i<6;i++)
        {
        for (j=0;j<6;j++)
            {
            if (k%6==0)
                printf("%d:",5-k/6);
            printf("%c",tabuleiro[k]);
            k++;
            }
        printf("\n");
        }
    printf("  abcdef");

}

int main(void) {
    char casas[200];
    char *tabuleiro;
    int j;
    int jogadas;
    char *vencedor;

    gets(casas);

    //criar tabuleiro
    tabuleiro=criar_tabuleiro();

    // Verificar a posição inicial dos dois reis
    if (verificar_posicao(separar_casas(casas)[0])==0||(verificar_p
osicao(separar_casas(casas)[1])==0))
```

```
        printf("Posicao Invalida.");
    else
    {
        inserir_pecas(casas,tabuleiro);
        jogadas=mover_pecas(casas,tabuleiro);
        imprimir_output(tabuleiro);
        if (jogadas%2==0)
            vencedor="pretas";
        else
            vencedor="brancas";

        printf("\nGanham as %s. Partida com %d jogadas validas.",ve
ncedor,jogadas);
    }

    return 0;
}
```

ANEXO II – Testes

Entrada	Saída	Fim por:
d1 d8 a2 b2 c2 d2 e2 f2 a4 b4 c4 d4 e4 f4 c2 c3 d4 c3 c3 c4	Posicao Invalida.	Rei inválido
d1 d5 a2 b2 c2 d2 e2 f2 a4 b4 c4 d4 e4 f4 c2 d2	5:...r.. 4:pppppp 3:..... 2:PPPPPP 1:...R.. abcdef Ganham as pretas. Partida com 0 jogadas validas.	Movimento invalido
d1 d5 a2 b2 c2 d2 e2 f2 a4 b4 c4 d4 e4 f4 c2 c3 0	5:...r.. 4:pppppp 3:..P... 2:PP.PPP 1:...R.. abcdef Ganham as brancas. Partida com 1 jogadas validas.	Movimento invalido
d1 d5 a2 b2 c2 d2 e2 f2 a4 b4 c4 d4 e4 f4 c2 c3 d4 c3 d1 c2 c3 b2 c2 b2 d5 d4 d2 d3	5:..... 4:ppprpp 3:...P.. 2:PR..PP 1:..... abcdef Ganham as brancas. Partida com 7 jogadas validas.	Movimento invalido
d1 d5 a2 b2 c2 d2 e2 f2 a4 b4 c4 d4 e4 f4 c2 c3 d4 c3 d1 c2 c3 b2 c2 b2 d5 d4 d2 d4	5:..... 4:ppprpp 3:..... 2:PR.PPP 1:..... abcdef Ganham as pretas. Partida com 6 jogadas validas.	Movimento invalido
d1 d5 a2 b2 c2 d2 e2 f2 a4 b4 c4 d4 e4 f4 c2 c3 d4 c3 d1 c2 c3 b2 c2 b2 d5 d4 d2 d3 d4 d3 e2 e3 d3 e3 b2 c2 e3 d3 c2 c1 d3 d2 c1 d2	5:..... 4:ppp.pp 3:..... 2:P..R.P 1:..... abcdef Ganham as brancas. Partida com 15 jogadas validas.	Tomada de rei

d1 d5 a2 b2 c2 d2 e2 f2 a4 b4 c4 d4 e4 f4 c2 c3 d4 c3 f2 f3 c3 d4	5:...r.. 4:ppp.pp 3:..p..P 2:PP.PP. 1:...R.. abcdef Ganham as brancas. Partida com 3 jogadas validas.	Movimento invalido
d1 d5 a8 b8 c9 d9 g2 g2 j4 i4 i4 s4 s4 z4 c2 c3 d4 c3 f2 f3 c3 d4	5:...r.. 4:..... 3:..... 2:..... 1:...R.. abcdef Ganham as pretas. Partida com 0 jogadas validas.	Sem entrada de peões
d1 d5 a2 b2 c2 d2 e2 f2 a4 b4 c4 d4 e4 f4 c2 c3 d4 c3 d1 c2 c3 b2 c2 b2 d5 d4 d2 d3 d4 d3 e2 e3 d3 e3 b2 c2 e3 d3 c2 c3 f2 f3 e4 f3 c3 c2 a2 a3 b4 a3 c2 d3	5:..... 4:p.p..p 3:p..R.p 2:..... 1:..... abcdef Ganham as brancas. Partida com 19 jogadas validas.	Tomada de rei
d1 d5 a2 b2 c2 d2 e2 f2 a4 b4 c4 d4 e4 f4 c2 c3 d4	5:...r.. 4:pppppp 3:..P... 2:PP.PPP 1:...R.. abcdef Ganham as brancas. Partida com 1 jogadas validas.	Movimento invalido