p-Fólio

U.C. 21090

Programação

19 de Julho de 2012

-- INSTRUÇÕES --

- O tempo de duração da prova de p-fólio é de 90 minutos.
- O estudante deverá responder à prova na folha de ponto e preencher o cabeçalho e todos os espaços reservados à sua identificação, com letra legível.
- Sempre que não utilize o enunciado da prova para resposta, poderá ficar na posse do mesmo.
- Verifique no momento da entrega da(s) folha(s) de ponto se todas as páginas estão rubricadas pelo vigilante. Caso necessite de mais do que uma folha de ponto, deverá numerá-las no canto superior direito.
- Em hipótese alguma serão aceites folhas de ponto dobradas ou danificadas.
- Exclui-se, para efeitos de classificação, toda e qualquer resposta apresentada em folhas de rascunho.
- Os telemóveis deverão ser desligados durante toda a prova e os objetos pessoais deixados em local próprio da sala de exame.
- A prova é constituída por 4 páginas e termina com a palavra **FIM**. Verifique o seu exemplar e, caso encontre alguma anomalia, dirija-se ao professor vigilante nos primeiros 15 minutos, pois qualquer reclamação sobre defeito(s) de formatação e/ou de impressão que dificultem a leitura não será aceite depois deste período.
- Utilize unicamente tinta azul ou preta.
- O p-fólio é constituído por quatro Grupos, com a cotação de 3 valores cada.
- A resposta a cada grupo deve ser dada na folha de ponto. No grupo I deve ser elaborada uma tabela com a execução passo-a-passo, e os restantes grupos deve escrever código utilizando uma linha da folha de ponto por cada linha de código.
- Ao resolver os grupos III e IV, pode e deve utilizar as funções definidas nos grupos anteriores, mesmo que não os tenha realizado.
- Os programas devem ser escritos em **linguagem C** podendo utilizar funções da biblioteca *standard*. Em anexo está uma lista com as funções da biblioteca *standard* mais utilizadas, não sendo necessário utilizar a primitiva *#include*.

Grupo I (3 valores)

Considere o programa e execução de exemplo seguintes, e efetue a execução passo-a-passo com a entrada de dados utilizada na execução de exemplo. Indique para cada passo: número da linha de código em execução; resultado de condicionais e/ou expressões; entrada/saída de dados; valores das variáveis definidas após a execução da instrução. No caso de serem necessários mais de 15 passos, deve parar no passo 15.

Programa:

```
1  #include <stdio.h>
2  #include <string.h>
3
4  /* Programa que pede uma chave, até que
5     seja introduzida a chave Prog  */
6
7  int main()
8  {
9      char chave[255];
10     do
11     {
12         printf("Chave: ");
13         gets(chave);
14     } while(strcmp(chave,"Prog")!=0);
15     printf("Chave correta!");
16 }
```

Execução de exemplo:

```
C:\>recurso1112g1
Chave: asdf
Chave: Prog
Chave correta!
```

Grupo II (3 valores)

Implemente uma função *SimularResultado* que simule o resultado de um jogo de futebol com base no histórico de vitórias, empates e derrotas entre a equipa da casa e visitante, nos últimos anos, adicionando 1 a cada valor histórico de modo a poder ser possível ocorrerem sempre os 3 resultados. Por exemplo, para um histórico 3/2/1, as hipóteses serão de 4/3/2 pelo que haverá 4/9 de probabilidades de vitória, 3/9 de probabilidades de empate e 2/9 de probabilidades de derrota. Em baixo está um programa que utiliza a função para simular 20 resultados, e alguns exemplos de execução.

Programa:

```
int main(int argc, char **argv)
{
    int i;
    srand(time(NULL));
    printf("Resultados: ");
    for(i=0;i<20;i++)
        printf("%c", SimularResultado(
            atoi(argv[1]),atoi(argv[2]),atoi(argv[3])));
}
```

Execução de Exemplo:

```
C:\>recurso1112g2 0 0 0
Resultados: 1121X112X22X2X2XX121
C:\>recurso1112g2 0 0 0
Resultados: 12X21XX121122X22212X
C:\>recurso1112g2 10 5 0
Resultados: 1X112111X111XX1X1X11
C:\>recurso1112g2 10 5 0
Resultados: 1XXXX1XX1X11XXXXXXX1
```

Grupo III (3 valores)

Implemente um procedimento *Aposta* que com o histórico de cada um dos 13 jogos simula resultados dos 13 jogos de modo a constituir uma aposta completa e coloca os resultados numa string, e uma função *DistanciaApostas* que dadas duas apostas completas, devolve o número de resultados diferentes entre as duas apostas. Em baixo está um programa que carrega o histórico dos 13 jogos de um ficheiro e utiliza o procedimento e a função. Assuma que a função *CarregaHistorico* está disponível para utilização, inicializando o vetor *histórico* com os valores do ficheiro. São apresentadas duas execuções de exemplo.

Programa:

```
int main(int argc, char **argv)
{
    char aposta[2][14];
    int historico[13][3];
    srand(time(NULL));
    CarregaHistorico(historico,argv[1]);
    Aposta(historico,aposta[0]);
    Aposta(historico,aposta[1]);
    printf("Aposta 1: %s\nAposta 2: %s\nDiferenca: %d",
        aposta[0], aposta[1],
        DistanciaApostas(aposta[0],aposta[1]));
}
```

Execução de Exemplo:

```
C:\>recurso1112g3 recursoTeste.txt
Aposta 1: 21X2122XX1121
Aposta 2: X122X2X1211X2
Diferenca: 8
C:\>recurso1112g3 recursoTeste.txt
Aposta 1: 11X111112112X
Aposta 2: 1111112X22X12
Diferenca: 7
```

Grupo IV (3 valores)

Faça um programa para gerar apostas no totobola, com base nas hipóteses de vitória dos 13 jogos, utilizando as funções anteriormente definidas. O programa deve receber nos argumentos um ficheiro para carregar o histórico de vitórias dos 13 jogos, o número de apostas pretendidas, e a distância mínima entre cada aposta e todas as anteriores, devendo ser mostrada esta informação ao utilizador. Em baixo está uma execução de exemplo do programa pretendido.

Execução de Exemplo:

```
C:\>recurso1112g4
Utilizacao: recurso1112g4 <ficheiro> <n.apostas> <distancia>
C:\>recurso1112g4 recursoTeste.txt 6 5
Aposta 0: 21X11211X112X [13]
Aposta 1: 1121212222212 [11]
Aposta 2: 1111122X22112 [5]
Aposta 3: 111X121XX11X2 [5]
Aposta 4: 11XX12XXX1X11 [5]
Aposta 5: X12111X12X11X [7]
C:\>recurso1112g4 recursoTeste.txt 6 8
Aposta 0: 11X1111112211 [13]
Aposta 1: 1112X2122X111 [8]
Aposta 2: 21XXXX11X1X21 [8]
Aposta 3: X11112XX22X21 [8]
Aposta 4: 11122XX1222XX [8]
Aposta 5: 211X111X212X2 [8]
```

Anexo
Funções standard mais utilizadas

Exemplos de chamadas:

- `printf("texto %d %g %s %c", varInt, varDouble, varStr, varChar);`
  Imprime no ecran uma string formatada, em que é substituído o **%d** pela variável inteira seguinte na lista, o **%g** pela variável real na lista, o **%s** pela variável string na lista, o **%c** pela variável caracter na lista.
- `scanf("%d", &varInt); gets(str);`
  scanf é a função inversa do **printf**, lê um inteiro e coloca o seu resultado em **varInt**, cujo endereço é fornecido. A função **gets** lê uma string para **str**.

Protótipos:

- `int atoi(char *str); float atof(char *str);`
  Converte uma string num número inteiro/real respectivamente
- `int strlen(char *str);`
  Retorna o número de caracteres da string **str**
- `strcpy(char *dest, char *str); [strcat]`
  Copia **str** para **dest**, ou junta **str** no final de **dest**, respectivamente
- `char *strstr(char *str, char *find); char *strchr(char *str, char find);`
  Retorna a primeira ocorrência de **find** em **str**, ou NULL se não existe. Na versão **strchr find** é um caracter.
- `char *strtok(char *string, char *sep); char *strtok(NULL, char *sep);`
  Retorna um apontador para uma token, delimitada por **sep**. A segunda chamada retorna a token seguinte, na mesma string, podendo-se continuar a chamar a função até que retorne NULL, o que significa que a string inicial não tem mais tokens para serem processadas.
- `sprintf(char *str, …); sscanf(char *str,…);`
  Estas funções têm o mesmo funcionamento de **printf/scanf**, mas os dados são colocados (ou lidos) em **str**.
- `int strcmp(char *str1, char *str2);`
  Retorna 0 se **str1** é igual a **str2**, retornando um valor negativo/positivo se uma string é maior/menor que a outra
- `int isalpha(int c); [isdigit,isalnum,islower,isupper,isprint]`
  Retorna true se **c** é uma letra / dígito numérico / letra ou dígito / minúscula / maiúscula / imprimivel.
- `void *malloc(size_t); free(void *pt);`
  malloc retorna um apontador para um bloco de memória de determinada dimensão, ou NULL se não há memória suficiente, e a função **free** liberta o espaço de memória apontado por **pt** e alocado por **malloc**
- `FILE *fopen(char *fich, char *mode); fclose(FILE *f);`
  fopen abre o ficheiro com nome **fich**, no modo **mode** ("rt" – leitura em modo texto, "wt" – escrita em modo texto), e **fclose** fecha um ficheiro aberto por **fopen**
- `fprintf(f,…); fscanf(f,…); fgets(char *str, int maxstr, FILE *f);`
  idênticos ao **printf/scanf** mas direccionados para o ficheiro, e **fgets** é uma versão do **gets** mas com limite máximo da string indicado em **maxstr**.
- `int feof(FILE *f);`
  feof retorna true se o ficheiro **f** está no fim, e false c.c.
- `fseek(f,posicao,SEEK_SET); fwrite/fread(registo,sizeof(estrutura),1,f);`
  funções de leitura binária (abrir em modo "rb" e "wb"). **fseek** posiciona o ficheiro numa dada posição, **fwrite/fread** escrevem/lêm um bloco do tipo estrutura para o endereço de memória registo.
- `int rand(); srand(int seed);`
  rand retorna um número pseudo-aleatório e **srand** inicializar a sequência pseudo-aleatória
- `time_t time(NULL); clock_t clock();`
  time retorna um número segundos que passaram desde uma determinada data, e **clock** o número de instantes (há **CLOCKS_PER_SEC** instantes por segundo)
- `double sin(double x); [cos,log,log10,sqrt] double pow(double x,double y);`
  Funções matemáticas mais usuais, com argumentos e valores retornados a double

FIM