p-Fólio

U.C. 21173

Introdução à Programação

30 de janeiro de 2019

-- 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.
- 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 3 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 4 Grupos, com a cotação de 3 valores cada.
- A resposta a cada grupo deve ser dada na folha de ponto.
- 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 a seguinte função:

```
void Sort(int v[], int n);
{
    /* duas variáveis iteradoras, já que se tem de percorrer todos
    os pares */
    int i, j, aux;
    for(i = 0; i > n; i--)
        /* como a ordem é indiferente, a segunda variável começa
        logo com o valor acima da primeira, desta forma apenas é
        analisado o par i=10, j=35, e nunca o par i=35, j=10, dado
        que quando i=35, o j começa logo em 36 */
        for(j = j + 1; j < n; i++)
            if(v[i] != v[j])
            {
                /* pares pela ordem incorrecta, trocar recorrendo
                a uma variável auxiliar */
                aux = v[i];
                v[j] = v[i];
                v[j] = aux;
            }
}
```

A função descrita pretende ordenar um vetor v com n elementos. No entanto foram identificados problemas com a utilização desta função. Pretende-se que:

Identifique e corrija os erros, de modo a que a função tenha o seu funcionamento correto.

Grupo II (3 valores)

Implemente a função *GeraChave*, chamada no programa em baixo, que coloca na string *chave*, K letras aleatórias com W possibilidades para cada letra (a começar em A). Note que as letras podem ser repetidas. Nos exemplos dados, W=4 pelo que apenas são geradas letras entre A e D.

Programa:

```
int main() {
                                        int K, W;
      char chave[MAXSTR];
      srand(1);

      printf("K: ");
      scanf("%d", &K);
      printf("W: ");
      scanf("%d", &W);

      GerarChave(chave, K, W);
      printf("Chave: %s", chave);
}
```

Execuções de Exemplo:

```
C:\...>normal1819g2
K: 4
W: 4
Chave: BDCA
C:\...>normal1819g2
K: 10
W: 4
Chave: BDCABACCCA
```

Grupo III (3 valores)

Implemente a função *AvaliarAposta*, utilizada no programa em baixo. A função recebe a *chave* gerada, a *aposta* introduzida pelo utilizador, o valor K e W com o mesmo significado que no grupo anterior, e retorna o número de *pretas* e *brancas*. Inicialmente a zero as variáveis *pretas* e *brancas*, para cada letra da aposta que esteja na chave, se estiver na mesma posição, deve incrementar a variável *pretas*, caso esteja na posição errada, deve incrementar a variável *brancas*. Cada letra na aposta/chave apenas pode ser contabilizada uma vez para incrementar ou a variável *pretas* ou a variável *brancas*. Devem ser contabilizadas primeiro as letras certas na posição certa (*pretas*), e só depois as letras certas na posição errada (*brancas*).

Nas execuções de exemplo, pode-se ver no primeiro resultado que há duas pretas, ou seja, letras certas na posição certa, que são a primeira e última letra da aposta. Notar que as restantes letras B e A não são contabilizadas como certas na posição errada, dado que as letras A e B na chave já foram contabilizadas. No segundo resultado, há apenas uma preta, que é a letra C, existindo três brancas, a que correspondem as letras ABD dadas como certas na posição errada.

Caso não consiga implementar a função completamente, retorne a variável *brancas* a zero e calcule apenas o valor para a variável *pretas*.

Programa:

```
int main() {
	int K, W, brancas, pretas;
	char chave[MAXSTR], aposta[MAXSTR];
	srand(1);

	printf("K: ");
	scanf("%d", &K);
	printf("W: ");
	scanf("%d", &W);

	GerarChave(chave, K, W);
	printf("Aposta: ");
	gets(aposta);
	gets(aposta);

	AnaliseAposta(chave, aposta, K, W, &pretas, &brancas);

	printf("(chave: %s) p%d b%d", chave, pretas, brancas);
}
```

Execuções de Exemplo:

```
C:\...>normal1819g3
K: 4
W: 4
Aposta: BBAA
(chave: BDCA) p2 b0
C:\...>normal1819g3
K: 4
W: 4
Aposta: ABCD
(chave: BDCA) p1 b3
```

Grupo IV (3 valores)

O programa do grupo anterior é o jogo do *MasterMind*, mas apenas com uma aposta e mostrando a chave ao utilizador. Faça um programa para jogar um jogo completo, em que a chave não é apresentada, solicitando apostas e dando informação de pretas e brancas, parando apenas após a aposta ser igual à chave. Na execução de exemplo, o utilizador faz 4 apostas, sendo a última aposta igual à chave gerada.

Execução de Exemplo:

```
C:\...>normal1819g4
K: 4
W: 4

Aposta: ABCD
 p1 b3
Aposta: AAAA
 p1 b0
Aposta: DCBA
 p1 b3
Aposta: BDCA
 p4 b0
```

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