p-Fólio

U.C. 21090

Programação

9 de fevereiro de 2015

-- 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 **6** 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 devem ser dadas respostas para todos os espaços por preencher no enunciado, e nos 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)

a) [1 valor] Considere o seguinte programa:

```
 1 #include <stdio.h>
 2
 3 int main()
 4 {
 5   /* declaração de duas variáveis inteiras */
 6   int x=-31, y=17;
 7
 8   /* trocar o valor de x com y */
 9   z=x+y;
10   y=x-y;
11   x=z-y;
12
13   /* mostrar os valores em x e em y */
14   printf("x: %d, y: %d",x,y);
15 }
```

Complete os espaços vazios da execução passo a passo:

P.	L.	Instr.	Resultado	
1	6	int ..		x=-31 , y= 17
2	9	x=x+y;		x=-14 , y= 17
3	10	y=x-y;		x=-14 , y=-31
4	11	x=x-y;		x= 17 , y= -31
5	14	printf..	x: 17, y:-31	z= 17 , y= -31

b) [1 valor] Considere a seguinte expressão lógica, que é verdadeira quando um ano é bissexto:

```
ano%4==0 && ano%100!=0 || ano%400==0
```

Indique para o ano 1612, quais as expressões avaliadas, e respectivo resultados parciais e final:

	Verdadeiro / Falso / Não Avaliado
ano%4==0	Verdadeiro
ano%100!=0	Verdadeiro
ano%400==0	~~FALSO~~ NÃO AVALIADO
Resultado	Verdadeiro

c) [1 valor] Considere o seguinte programa:

```
1 #include <stdio.h>
 2
 3 /* programa que calcula o próximo ano bissexto */
 4 int main()
 5 {
 6   int i, ano;
 7   printf("Indique Ano: ");
 8   scanf("%d", &ano);
 9   i=ano+1;
10   while((i%4!=0 || i%100==0) && i%400!=0)
11     i++;
12   printf("Proximo ano bissexto: %d",i);
13 }
```

E a seguinte execução:

```
c:\>prog1
Indique Ano: 2009
Proximo ano bissexto: 2012
```

Sendo conhecido o estado de um passo, numa execução passo a passo, preencha os valores relativos ao passo seguinte:

```
P.|L.|Instr.  |Resultado |
...
 6|11|i++     |          |i=2011, ano=2009
```

Passo: 7
Linha: 10
Variáveis: i=2011, ano=2009
Resultado:

● Verdadeiro	○ Falso	○ Sem expressão lógica

Grupo II (3 valores)

Implemente um procedimento *MostraInteiros* que recebe um vetor de inteiros, número de elementos no vetor, uma string, e mostra a string seguida dos inteiros. Em baixo está um programa que utiliza a função e uma execução de exemplo, simulando o lançamento de 5 dados.

Programa:

```
void main()
{
    int dados[5], i;
    srand(time(NULL));
    for(i = 0; i < 5; i++)
        dados[i] = 1 + rand() % 6;

    MostraInteiros(dados, 5, "Valor dos dados lancados: ");
}
```

Execução de Exemplo:

```
C:\...>normal1415g2
Valor dos dados lancados: 3 6 1 3 4
C:\...>normal1415g2
Valor dos dados lancados: 1 5 3 3 3
C:\...>normal1415g2
Valor dos dados lancados: 3 4 3 4 3
C:\...>normal1415g2
Valor dos dados lancados: 1 3 5 4 3
```

Grupo III (3 valores)

Implemente a função ValorDados, que recebe um vetor de 5 inteiros, com números de dados (inteiros de 1 a 6), e retorna o valor no poker de dados (não considerando a sequência). Deverá retornar: 0 no caso de não existir qualquer par (um par é um conjunto de números com dois iguais); 1 no caso de existir apenas um par; 2 se existirem dois pares; 3 se existir um trio; 4 se existir um par e um trio; 5 se existir um poker quadruplo (4 números iguais); 6 se existir um poker real (5 números iguais). Em baixo está o programa que utiliza a função pedida e uma execução de exemplo, onde se pode ver que no primeiro lançamento o resultado é 2 pares, porque o número 4 e 5 estão ambos repetidos. No terceiro lançamento houve um par e um trio, dado que o número 3 saiu 3 vezes, e o número 4 duas vezes.

Programa:

```
void main()
{
    int dados[5], i;
    char *resultados[]={
        "nada", "par", "2 pares", "trio", "par + trio",
        "poker quadruplo", "poker real"
    };
    srand(time(NULL));
    for(i = 0; i < 5; i++)
        dados[i] = 1 + rand() % 6;

    MostraInteiros(dados, 5, "Valor dos dados lancados: ");

    printf("\nResultado: %s",resultados[ValorDados(dados)]);
}
```

Execução de Exemplo:

```
C:\...>normal1415g3
Valor dos dados lancados: 5 4 5 4 3
Resultado: 2 pares
C:\...>normal1415g3
Valor dos dados lancados: 6 1 2 3 4
Resultado: nada
C:\...>normal1415g3
Valor dos dados lancados: 3 3 4 4 3
Resultado: par + trio
C:\...>normal1415g3
Valor dos dados lancados: 1 2 6 4 3
Resultado: nada
```

Grupo IV (3 valores)

Faça um programa que expanda o programa do grupo anterior, permitindo um segundo lançamento mas retendo os dados que o utilizador quiser. Deve após o primeiro lançamento, deve ser pedido quais os dados a reter. O utilizador digita uma string com 5 dígitos, um por cada dado. Se 0 o dado não é retido, se 1 o dado é retido. Em baixo encontra-se uma execução exemplo. Pode-se ver que no primeiro jogo saiu um par, e foram retidos os dados correspondentes ao par. Os restantes dados foram substituídos por lançamentos aleatórios, o que acabou por elevar o resultado para 2 pares. No segundo jogo saiu um trio, e no segundo lançamento elevou-se esse valor para um poker quadruplo.

Execução de Exemplo:

```
C:\...>normal1415g4
Valor dos dados lancados: 4 3 1 1 2
Resultado: par
Reter (10100 reter o primeiro e o dado central):00110
Valor dos dados lancados: 4 3 1 1 3
Resultado: 2 pares
C:\...>normal1415g4
Valor dos dados lancados: 5 2 2 3 2
Resultado: trio
Reter (10100 reter o primeiro e o dado central):01101
Valor dos dados lancados: 5 2 2 2 2
Resultado: poker quadruplo
C:\...>normal1415g4
Valor dos dados lancados: 3 6 1 2 1
Resultado: par
Reter (10100 reter o primeiro e o dado central):00101
Valor dos dados lancados: 2 4 1 3 1
Resultado: par
```

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