p-Fólio

U.C. 21090

Programação

11 de fevereiro de 2016

-- 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 **5** 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=205, y=-105;
 7
 8   /* trocar o valor de x com y */
 9   x=x+y;
10   y=x-y;
11   x=x-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=____, y=____
 2| 9|x=x+y;  |          |x=____, y=____
 3|10|y=x-y;  |          |x=____, y=____
 4|11|x=x-y;  |          |x=____, y=____
 5|14|printf..|x:50, y:52|x=____, y=____
```

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 704, quais as expressões avaliadas, e respectivo resultados parciais e final:

`ano%4==0`	○ Verdadeiro	○ Falso	○ Não avaliado
`ano%100!=0`	○ Verdadeiro	○ Falso	○ Não avaliado
`ano%400==0`	○ Verdadeiro	○ Falso	○ Não avaliado
`Resultado`	○ Verdadeiro	○ Falso	○ Não avaliado

c) [1 valor] Considere o seguinte programa:

```
 1 #include <stdio.h>
 2
 3 /* programa que calcula N^N */
 4 int main()
 5 {
 6   int i=0, N, resultado=1;
 7   printf("Indique N: ");
 8   scanf("%d", &N);
 9   while(i<N) {
10     resultado*=N;
11     i++;
12   }
13   printf("Resultado %d^%d: %d",N,N,resultado);
14 }
```

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

```
P.|L.|Instr.  |Resultado |
...
14|10|res*=.. |          |i=3, N=4, resultado=256

Passo: 15
Linha: _____
Variáveis: i=_____, N=_____, resultado=_____
Resultado:
```

○ Verdadeiro	○ Falso	○ Sem expressão lógica

Grupo II (3 valores)

Implemente um procedimento *CifraCesar* que recebe uma string e uma chave inteira, e aplica a Cifra de Júlio César aos dígitos numéricos (caracteres de 0 a 9). Esta cifra substitui cada carácter por K caracteres à frente, dando a volta tanto nas letras como nos dígitos, isto é, 0 com um K de 11, fica 1 (10 dígitos mais 1). Deve deixar inalterados os caracteres não numéricos. No exemplo pode-se ver que a chave 12, transforma o 1 em 3 (dá a volta e ainda incrementa duas unidades), o 2 em 4, e assim sucessivamente.

Programa:

```
int main(int argc, char** argv)
{
  char str[255];
  if(argc!=2)
    return;
  printf("\nNumero: ");
  gets(str);
  CifraCesar(str,atoi(argv[1]));
  printf("Codificacao: %s",str);
}
```

Execução de Exemplo:

```
C:\...>normal1516g2 12

Numero: 1234
Codificacao: 3456
```

Grupo III (3 valores)

Melhore agora o procedimento *CifraCesar*, de modo a aplicar a cifra tanto aos dígitos de acordo com o grupo II, como também às letras minúsculas, e às letras maiúsculas. Estes três conjuntos de caracteres devem ser considerados em separado, sendo um dígito codificado para um dígito, uma letra minúscula codificada para uma letra minúscula, e uma letra maiúscula para uma letra maiúscula. O programa que chama a função é idêntico ao do grupo II. Na execução de exemplo, pode-se ver a letra 'a' transformada para a letra 'm', que está 12 posições à frente, e o número no texto com a mesma codificação que no grupo II, tendo-se utilizado a mesma chave.

Execução de Exemplo:

```
C:\...>normal1516g3 12

Texto: abcd 1289 ABCD
Codificacao: mnop 3401 MNOP
```

Grupo IV (3 valores)

Faça um programa que receba nos argumentos o nome de um ficheiro a codificar, e retorne o ficheiro codificado para a consola. Assuma que o ficheiro não tem linhas superiores a 200 caracteres. Na execução de exemplo, o ficheiro com uma possível resolução deste grupo, é codificado com a chave 12435. Vê-se a parte inicial do resultado do programa..

Execução de Exemplo:

```
C:\...>normal1516g4 normal1516g4.c 12435
#pujsbkl <zakpv.o>
#pujsbkl <zakspi.o>

/*
Jpmyh kl JÚzhy:
N7 - hwluhz kÝnpavz
N8 - kÝnpavz l slayhz thp·zjbshz/tpu·zjbshz
N9 - jvkpmpjhy bt mpjolpyv
*/

johy Jpmyh(johy h, pua johcl, johy pupjpv, johy mpt)
{
```

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