p-Fólio

U.C. 21090

Programação

06 de Fevereiro de 2013

-- 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:

```
 3  int main()
 4  {
 5      int batalhoes=0, cavalaria=0, artilharia=0;
 6      printf("Numero de batalhoes: ");
 7      scanf("%d",&batalhoes);
 8      /* descontar nos batalhões a artilharia e cavalaria, se possível */
 9      if(batalhoes>0)
10      {
11          artilharia=batalhoes/10;
12          batalhoes-=artilharia*10;
13          cavalaria=batalhoes/5;
14          batalhoes-=cavalaria*5;
15      } else
16          batalhoes=0;
17      /* mostrar resultado, já que o número de tropas de infantaria
18         corresponde ao número de batalhões que não foram contabilizados
19         na arquilharia nem na cavalaria */
20      printf("%d Infantaria, %d Cavalaria, %d Artilharia",
21          batalhoes, cavalaria, artilharia);
22  }
```

Execução de exemplo:

```
C:\>prog1
Numero de batalhoes: 14
4 Infantaria, 0 Cavalaria, 1 Artilharia
```

Grupo II (3 valores)

Implemente uma função *RemoveMaisAlto* que recebe um conjunto de elementos inteiros positivos num vetor, e número de elementos, retornando o valor mais alto. O valor retornado deverá ser colocado a zero no vetor, de modo a que, se a função for chamada várias vezes sejam retornados os restantes números mais altos, como se pode verificar na execução do programa exemplo.

Programa:

```
void main()
{
    int i, vetor[] = {1, 45, 23, 4};
    for(i=0;i<4;i++)
        printf("\nMais alto: %d", RemoveMaisAlto(vetor,4));
}
```

Execução de Exemplo:

```
C:\>prog2
Mais alto: 45
Mais alto: 23
Mais alto: 4
Mais alto: 1
```

Grupo III (3 valores)

Implemente um procedimento *Empacota* que recebe um conjunto de elementos e tamanho do pacote, e empacota os elementos processando-os do mais alto para o mais baixo, colocando cada elemento no primeiro pacote com espaço disponível. O procedimento pode ter informação de *debug*. No exemplo, essa informação são as linhas de *Pacote 0 += 9*, a *Pacote 0 +=1*. Estas duas linhas significam que o elemento de valor nove e o elemento de valor um foram colocados no pacote 0. O procedimento deve retornar um vector, com o espaço livre em cada pacote. No programa exemplo, o vetor *pacote* fica com o espaço disponível em cada pacote, sendo o tamanho de cada pacote 10, tal como o número de elementos do vetor. Neste caso, o pacote 2 ficou com espaço livre 1, já que foram colocados os elementos 5 e 4, que ocuparam 9 de 10 unidades.

Programa:

```
void main()
{
    int elemento[10] = {1, 9, 3, 2, 2, 3, 4, 6, 5, 4};
    int pacote[10], i, nPacotes;
    nPacotes=Empacota(elemento,pacote,10,10);
    for(i=0;i<nPacotes;i++)
        printf("\nEspaco livre no pacote %d: %d", i, pacote[i]);
}
```

Execução de Exemplo:

```
C:\>prog3
 Pacote 0 += 9
 Pacote 1 += 6
 Pacote 2 += 5
 Pacote 1 += 4
 Pacote 2 += 4
 Pacote 3 += 3
 Pacote 3 += 3
 Pacote 3 += 2
 Pacote 3 += 2
 Pacote 0 += 1
Espaco livre no pacote 0: 0
Espaco livre no pacote 1: 0
Espaco livre no pacote 2: 1
Espaco livre no pacote 3: 0
```

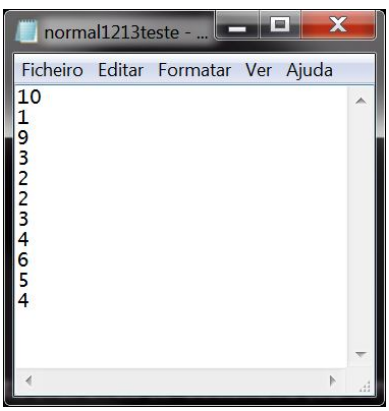
Grupo IV (3 valores)

Faça um programa que recebe um ficheiro com números, um número por linha, e um parâmetro com o tamanho do pacote, e retorna o resultado do empacotamento de forma idêntica ao grupo III. A primeira linha do ficheiro não é um elemento mas sim o número de elementos no resto do ficheiro. Na imagem é visível o conteúdo do ficheiro utilizado na execução de exemplo. Não deve assumir um limite máximo para o número de elementos no ficheiro.

Execução de Exemplo:

```
C:\>prog4
Utilizacao: prog4 <ficheiro> <tamanho>
C:\>prog4 normal1213teste.txt 10

 Pacote 0 += 9
 Pacote 1 += 6
 Pacote 2 += 5
 Pacote 1 += 4
 Pacote 2 += 4
 Pacote 3 += 3
 Pacote 3 += 3
 Pacote 3 += 2
 Pacote 3 += 2
 Pacote 0 += 1
Espaco livre no pacote 0: 0
Espaco livre no pacote 1: 0
Espaco livre no pacote 2: 1
Espaco livre no pacote 3: 0
```



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