Exercícios – Conceitos base de C

Esta folha de exercícios tem como objetivo recapitular os conceitos adquiridos em Programação 1 e que serão uma base essencial em Programação 2. Alguns exercícios foram apresentados no primeiro semestre 2017/18 e inseridos em minitestes passados. São ainda propostos 5 problemas complementares.

1 - Implemente uma função, com parâmetros *a* e *x,* para o cálculo de $f(x) = ax^2$ (parábola). Utilize a função num programa que apresenta os valores de *f(x),* para valores de *x* num determinado intervalo definido pelo utilizador. O utilizador deve especificar os limites (inferior e superior) do intervalo, bem como o incremento a utilizar.

Exemplo

```
Qual o valor de a? 2
Qual o intervalo? 1 2
Qual o incremento? 0.5
f(1.0)=2.0
f(1.5)=4.5
f(2.0)=8.0
```

2 – Escreva um programa que preencha cada posição de dois vetores (de 5 elementos inteiros não repetidos cada) com números introduzidos pelo utilizador e depois imprima os que são comuns a ambos os vetores.

Exemplo

```
Vetor 1:
Introduza 1º número: 5
Introduza 2º número: 3
Introduza 3º número: 6
Introduza 4º número: 9
Introduza 5º número: 10
Vetor 2:
Introduza 1º número: 1
Introduza 2º número: 7
Introduza 3º número: 3
Introduza 4º número: 8
Introduza 5º número: 6

Comuns: 3 6
```

3 – Escreva um programa que preencha cada posição de um vetor (de 10 elementos reais) com 10 números introduzidos pelo utilizador. Deverá depois imprimir:

a. A média de todos os seus elementos, calculada por uma função

```
float avg(float x[]);
```

b. O maior elemento, calculado por uma função

```
float max(float x[]);
```

c. O menor elemento, calculado por uma função

```
float min(float x[]);
```

d. O conteúdo do vetor;

Exemplo

```
Introduza 1º número: 5.1
Introduza 2º número: 3.2
Introduza 3º número: 6.3
Introduza 4º número: 9.4
Introduza 5º número: 10.9
Introduza 6º número: 3.8
Introduza 7º número: 1.7
Introduza 8º número: -1.5
Introduza 9º número: 1.0
Introduza 10º número:9.2
Média: 4.91
Máximo: 10.9
Mínimo: -1.5
Vetor: 5.1 3.2 6.3 9.4 10.9 3.8 1.7 -1.5 1.0 9.2
```

4 – Escreva uma função

```
int ordenado(int src[],int n);
```

que verifica se os n primeiros elementos do vetor `src` estão ordenados por ordem crescente. A função deve retornar 1 em caso afirmativo e 0 caso contrário.

Escreva um programa que leia uma sequência de N números inteiros para um vetor, com N especificado pelo utilizador. O programa não deve permitir a introdução de sequências com mais do que 100 números. Utilizando a função `ordenado` verifique se os valores introduzidos estão por ordem crescente. Em caso afirmativo imprima o vetor por ordem decrescente .

Exemplo

```
Introduza o número de números: 7
Introduza 1º número: 1
Introduza 2º número: 3
Introduza 3º número: 5
Introduza 4º número: 7
Introduza 5º número: 8
Introduza 6º número: 9
Introduza 7º número: 10

O vetor está ordenado!
Elementos do vetor por ordem decrescente:
{10,9,8,7,5,3,1}
```

5 – Escreva um procedimento

```
void inverte(char *strOriginal, char *strInvertida);
```

que inverte a ordem dos caracteres de uma *string*. No primeiro argumento do procedimento é passada a *string* original e no segundo argumento é devolvida a *string* invertida.

Utilize este procedimento num programa que determina se uma palavra introduzida pelo utilizador é capicua. O programa deve permitir ao utilizador testar o número de strings que pretender e apenas termina quando o utilizador introduz "." (ponto final).

Exemplo

```
Palavra? Programa
Resultado: programa nao e capicua.
Palavra? ana
Resultado: ana e capicua.
```

6 – Escreva um programa que pede ao utilizador para escrever uma frase e apresenta no ecrã quantas palavras constituem a frase, a palavra de maior comprimento e o comprimento médio das palavras. Para simplificar assuma que as palavras são separadas por um único carácter de espaço.

Exemplo

```
Frase? O jornal de hoje tem na capa uma fotografia interessante
Numero de palavras: 10
Palavra maior: interessante
Comprimento medio: 4.7
```

7 – Considere uma turma com n estudantes (n <= 100). Pretende-se que implemente um programa que calcule as notas finais da turma, da seguinte forma: comece por ler, por esta ordem, a nota teórica (NT) e a nota prática (NP) de cada estudante (a leitura deve terminar quando a nota teórica lida for -1). Armazene num vetor a nota final de cada estudante, sabendo que a NT vale 60% da nota final e a NP vale 40% dessa nota. Calcule a média dos valores no vetor, sem alterar o seu conteúdo. Considere que, para efeitos de cálculo da média, todas as notas entre 9.3 e 9.5 devem ser contabilizadas como 9.5. Implemente e utilize no seu programa as seguintes funções:

- ```
  int ler_notas (float * notas);
  /* lê a NT e a NP, para cada estudante, e guarda no vetor notas a
  correspondente nota final; retorna o número de valores efectivamente
  guardados */
  ```
- ```
  float media (float * notas, int n);
  /* calcula e retorna a media das notas da turma (armazenadas no vetor
  notas, de n elementos), atendendo ao descrito acima */
  ```

O seu programa pode ser testado com o ficheiro ***notas.txt*** [exemplo de utilização: ./ex7 < notas.txt]. Para esse ficheiro o resultado deverá ser:

```
A media das notas da turma foi 12.48
```

8 – Numa linha de enchimento de garrafas é mantido um registo ao longo do dia do volume colocado em cada garrafa. Sabendo que por dia não são enchidas mais do que 1000 garrafas, escreva um programa que lê e guarda num vetor os volumes de enchimento realizados ao longo de um dia com as seguintes características:

- O número de valores lidos é especificado pelo utilizador.
- Pretende-se determinar o volume médio de enchimento, sendo que deve ser excluído do cálculo do valor médio os valores que se encontram acima de um valor máximo e abaixo de um valor mínimo (ambos a especificar pelo utilizador).

Implemente e utilize no seu programa as seguintes funções:

- ```
  int ler_volumes(float *volumes);
  /* lê e guarda no vetor volumes os valores de enchimento de um dia;
  retorna o número de valores lidos */
  ```
- ```
  float volume_medio(float *volumes, int nvolumes, float max, float min);
  /* determina o volume médio de enchimento */
  ```

O programa pode ser testado com o ficheiro ***volumes.txt*** [exemplo de utilização: ./ex8 < volumes.txt]. Para esse ficheiro o resultado deverá ser:

```
Foram lidos 100 valores.
Indique valor mínimo: 0.2
Indique valor máximo: 0.8
O volume medio de enchimento foi de 0.504 litros.
```

9 – Um colecionador de álbuns musicais pretende manter um registo de todos os álbuns que guarda na sua coleção. Para cada álbum é necessário guardar informação diversa. Implemente um programa que utilize as seguintes declarações:

```
#define NALBUNS 100

typedef struct{
   char artista[100];
   char titulo[100];
   int ano;
   int formato;    /* 1=vinil, 2=cd, 3=dvd */
} item;

int ler_albuns(item coleccao[]);
void lista_albuns(item coleccao[], int n);
int albuns_artista(item coleccao[], int n, char *artista, int formato);
```

Considere ainda que:

- A função `ler_albuns` preenche o vetor `coleccao` com informação introduzida pelo utilizador e retorna o número de álbuns efectivamente lidos. Para cada álbum devem ser lidos 4 valores: nome do artista (composto por duas palavras), título (uma só palavra), ano e formato (1, 2 ou 3). A leitura deverá terminar quando não for possível ler todos os dados de um álbum. Sugestão: verifique o valor de retorno da função scanf. Esta função deve ainda garantir que não são introduzidos mais do que `NALBUNS`.
- A função `albuns_artista` retorna o número de álbuns de um determinado artista, num determinado formato. No caso do valor do parâmetro `formato` ser igual 0, deverá ser retornado o número total de álbuns desse artista, independentemente do formato. Considere a estrutura do programa inicial para testar a função que desenvolveu.

O seu programa pode ser testado com o ficheiro ***albuns.txt*** [exemplo de utilização: ./ex9 < albuns.txt]. Para esse ficheiro o resultado deverá ser:

```
Lista carregada com 50 albuns.
Existem 12 albuns em vinil do artista Sergio Godinho.
Existem 18 albuns no total do artista Sergio Godinho.
```

10 – Desenvolva um programa para ajudar na gestão de tarefas diárias. Para o efeito pretende-se registar. Implemente um programa que utilize as seguintes declarações:

```
#define NTODOS 100
#define NCHARS 30

typedef struct {
  char nome[NCHARS];
  char tipo[NCHARS];
  int duracao;
  int diaConclusao;
} tarefa;

int nova_tarefa (tarefa *nova);
int ler_tarefas (tarefa lista[]);
void lista_tarefas (tarefa lista[], int n);
float analisa_tarefas (tarefa lista[], int n, tarefa *maisLonga);
```

Considere ainda que:

- A função `nova_tarefa` permite ler uma nova tarefa com informação introduzida pelo utilizador, que apenas será válida quando for possível ler todos os dados de uma tarefa. Cada tarefa é definida pelo respetivo nome (uma palavra), o seu tipo (uma palavra, por exemplo: pessoal, ensino, desporto, etc.), o tempo estimado (em horas) para a sua

realização e o dia do mês em que deve estar concluída. A função retorna 1 se consegue ler toda a informação de uma tarefa e 0 caso não consiga. Sugestão: verifique o valor de retorno da função scanf.

- A função `ler_tarefas` preenche o vetor `lista` recorrendo à função `nova_tarefa`. Deverá ler um conjunto sucessivo de tarefas e retornar o número de tarefas lidas. Esta função deve ainda garantir que não são introduzidos mais do que `NTODOS`.
- A função `analisa_tarefas` analisa a lista de tarefas e devolve a média de durações por retorno e a tarefa com maior duração na variável `maisLonga`, que é passada por referência à função.

O seu programa pode ser testado com o ficheiro ***todosJaneiro.txt*** [exemplo de utilização: ./ex10 < todosJaneiro.txt]. Para esse ficheiro o resultado deverá ser:

```
Número de tarefas lidas: 12
Duracao media das tarefas: 4.9 horas
Tarefas mais longa: Estudar_PROG1
```

11 (complementares) – Existem diversas competições nacionais e internacionais de programação onde são propostos desafios de programação e algoritmia interessantes com diferentes níveis de complexidade. Na página "Sphere online judge" podem ser encontrados alguns desses desafios. Neste exercício são propostos os seguintes problemas: ADDREV, HELLOKIT, TOANDFRO, BITMAP e STPAR – que podem ser acedidos através do URL:

http://www.spoj.com/problems/<nome_do_exercicio>