GUIÃO 02 – INTRODUÇÃO À LINGUAGEM DE PROGRAMAÇÃO C

A documentação da linguagem C pode ser consultada em https://en.cppreference.com/w/c

1. Analise o programa do ficheiro **`integer_arithmetic_pitfalls.c`** que efetua uma sequência de operações aritméticas simples.

Sem executar o programa, tome nota do output esperado.

Compile e execute o programa. O output que previu está correto? Porquê?

Compile novamente o programa, usando a *flag* de compilação **`-Wsign-compare`** ou a *flag* de compilação **`-Wextra`**, e analise os avisos de compilação adicionais.

Qual é o maior valor inteiro, do tipo **`int`**, representável no seu computador? E o menor?

Imprima os valores das constantes **`INT_MAX`** e **`INT_MIN`** definidas no ficheiro cabeçalho **`limits.h`**.

2. O progama do ficheiro **`primes.c`** implementa uma versão do Crivo de Eratóstenes, mas contém **erros**.

Compile e execute o programa. Analise o output obtido e o código para corrigir esses erros.

Sugestão: em Linux pode usar o programa **`valgrind`** para verificar a existência de eventuais erros de gestão de memória (*memory leaks*).

3. As funcionalidades da *Strings Library* estão definidas no ficheiro cabeçalho **`string.h`**.

Usando as **funcionalidades disponibilizadas** nessa biblioteca, escreva um programa que, após ler duas *strings*:

- Conte os caracteres da primeira *string* que são letras do alfabeto.
- Conte os caracteres da segunda *string* que são letras maiúsculas.
- Converta todas as letras maiúsculas, das duas *strings*, para minúsculas.
- Compare as duas *strings* resultantes e escreva uma mensagem indicando que são iguais, ou apresentando as duas *strings* na sua ordem lexicográfica.
- Crie uma cópia da segunda *string*.
- Crie e imprima uma *string* que resulta da concatenação da segunda *string* com a sua cópia.

4. Desenvolva uma função que efetua a permutação circular dos valores de três variáveis inteiras:

```
void Permute(int* a, int* b, int* c)
```

Desenvolva um simples programa de teste que imprima os valores das variáveis inteiras antes e depois da sua permutação circular.

5. Desenvolva e teste as três funções seguintes que operam sobre *arrays* de números reais.

```
// Display the contents of array a with n elements
// Pre-Conditions: a != NULL and n > 0
// Example of produced output: Array = [ 1.00, 2.00, 3.00 ]
void DisplayArray(double* a, size_t n) { ... }

// Read the number of elements, allocate the array and read its elements
// Condition: number of elements > 0
// Pre-Condition: size_p != NULL
// Return NULL if memory allocation fails
// Set *size_p to ZERO if memory allocation fails
double* ReadArray(size_t* size_p) { ... }

// Allocate and return a new array with (size_1 + size_2) elements
// which stores the elements of array_1 followed by the elements of array_2
// Pre-Conditions: array_1 != NULL and array_2 != NULL
// Pre-Conditions: size_1 > 0 and size_2 > 0
// Return NULL if memory allocation fails
double* Append(double* array_1, size_t size_1, double* array_2, size_t size_2)
{ ... }

// Test example:    Array = [ 1.00, 2.00, 3.00 ]
//                  Array = [ 4.00, 5.00, 6.00, 7.00 ]
//                  Array = [ 1.00, 2.00, 3.00, 4.00, 5.00, 6.00, 7.00 ]
```

ATENÇÃO: no final do programa de teste, não se esqueça de libertar a memória alocada dinamicamente.

**** Exercícios Adicionais ****

6. O código do algoritmo de pesquisa binária contido no ficheiro **`binary_search.c`** contém **erros**.

Compile e execute o programa. Analise o output obtido e o código para corrigir esses erros.

Sugestão: familiarize-se com o *debugger* **`gdb`** e use-o para executar o código linha-a-linha e inspecionar o valor de algumas variáveis.

7. É possível representar **polinómios de grau n** usando um ***array* de (n + 1) elementos**, em que o elemento de índice zero é o coeficiente do termo de maior grau e o elemento de índice n é o coeficiente do termo de grau zero.

Desenvolva e teste as funções seguintes que operam sobre polinómios representados como *arrays* de números reais (**`double`**).

```
// The coefficients of a degree n polynomial
// are stored in an array p of size (n + 1)
// p[0] is the coefficient of the largest degree term
// p[n] is the coefficient of the zero-degree term


// Display a polynomial
// Pre-Conditions: coef != NULL and degree >= 0
// Example of produced output:
// Pol(x) = 1.000000 * x^2 + 4.000000 * x^1 + 1.000000
void DisplayPol(double* coef, size_t degree) { ... }
```

```
// Compute the value of a polynomial using Horner's method:
// Pre-Conditions: coef != NULL and degree >= 0
double ComputePol(double* coef, size_t degree, double x) { ... }

// Test example:
// Pol(x) = 1.000000 * x^2 + 4.000000 * x^1 + 1.000000
// Pol(2.000000) = 13.000000


// Compute the real roots, if any, of a second-degree polynomial
// Pre-Conditions: coef != NULL and degree == 2 and coef[0] != 0
// Pre-Conditions: root_1 != NULL and root_2 != NULL
// Return values:    0 -> no real roots
//                   1 -> 1 real root with multiplicity 2
//                   2 -> 2 distinct real roots
// The computed root values are returned via the root_1 and root_2
// pointer arguments
// Assign 0.0 to the *root_1 and *root_2 if there are no real roots
unsigned int
GetRealRoots(double* coef, size_t degree, double* root_1, double* root_2)
{ ... }
```