Exercícios de programação em C - III

Ligações úteis:

- Slides de *Visão geral da linguagem C* disponibilizados no moodle;
- Apontamentos de Programação Imperativa (CC1003), em particular:
  - Apontadores;
  - Programação com apontadores;
- The C Book;
- Everything you need to know about pointers in C;

---

1. O seguinte fragmento de código, que tem por objetivo ler da entrada padrão duas sequências de inteiros de tamanho **n** (um inteiro por linha), e proceder à soma desses arrays guardando o resultado num novo array, apresenta um problema muito grave:

```c
#include <stdio.h>
#include <stdlib.h>

int* readarray(int n){
    int i;
    int v[n];
    for (i=0; i<n; i++){
        scanf("%d\n", v+i)
    }
    return v;
}

int* somaarrays(int *a, int *b, int n) {
    /* A completar:
        deverá retornar um novo array
        com a soma dos arrays arrays a e b,
        elemento a elemento
    */
}
int printarray(int *v, int n){
    // A completar:
    // deverá imprimir os elementos de v
}


int main(){
    int n;
    int *va, *vb, *vr;
    scanf("%d\n", &n);
    va = readarray(n);
    vb = readarray(n)
```

```
    vr = somaarrays(a, b, n);
    printarray(va, n);
    printarray(vb, n);
    printarray(vr, n);
    return 0;
}
```

- Qual é esse problema?
- Complete o código do programa, corrigindo o problema identificado. Compile e teste o programa.

2. *Frações*

- Defina através de um `typedef` uma estrutura `FRAC` que seja adequada para conter uma fração com numerador e denominador inteiros. Deve ser incluída a seguinte informação: sinal, numerador (positivo), denominador (positivo), indicação de erro (e.g., devido a divisão por zero). Por exemplo, -1/4 é representado por:

  Sinal : -1, Num: 1, Den: 4, Erro: 0 (não tem erro)

- Defina uma função com o protótipo

  ```
  FRAC simp(FRAC f);
  ```

  que retorna o argumento simplificado.

- Defina uma função com o protótipo

  ```
  FRAC soma(FRAC, FRAC);
  ```

  que retorna a soma dos dois argumentos.

- Defina uma função com o protótipo

  ```
  FRAC sub(FRAC, FRAC);
  ```

  que retorna a diferença dos dois argumentos.

- Defina uma função com o protótipo

  ```
  FRAC mult(FRAC, FRAC);
  ```

  que retorna o produto dos dois argumentos.

- Defina uma função com o protótipo

  ```
  FRAC div(FRAC, FRAC);
  ```

  que retorna o quociente do primeiro argumento pelo segundo.

- Escreva um programa que permita testar as funções definidas anteriormente, da seguinte forma: Existe uma fração inicial **resultado**, com o valor 0. O utilizador dá um dos seguintes comandos:
  - **+**: o utilizador dá uma fração que é adicionada a `resultado`.

- **-**: o utilizador dá uma fração que é subtraída a `resultado`.
- ***:** o utilizador dá uma fração que é multiplicada pela fração em `resultado`.
- **/**: o utilizador dá uma fração pela qual a fração em `resultado` será dividido.

Cada uma das operações anteriores deverá apresentar o resultado simplificado.

3. Um vector `lista[]` de nomes e telefones, tem a seguinte estrutura, e é inicializada da forma indicada:

```
#define MAX_LISTA 1000
struct pessoa{
        char* nome;
        char* telefone;
};
typedef struct pessoa PESSOA;

PESSOA lista[MAX_LISTA] = {{"rui", "226664441"},
                           {"ana", "214444444"}};
int n = 2;  // Número de pessoas da lista
```

- Escreva uma função com o seguinte protótipo, que pesquise um nome **pal** na lista, indicando o índice onde se encontra, ou **-1** caso não exista.

```
int pesquisa(char *pal);
```

- Escreva uma função semelhante à da alínea anterior, mas que só procura nome **pal** apartir do índice **i** (parâmetro da função) até o fim da lista.

```
int pesqui(int i, char *pal);
```

- Escreva um programa que, utilizando a função `pesqui()`, leia repetidamente um nome e imprima os telefones correspondentes ou "**não existe!**".
- Altere o programa para aceitar os seguintes comandos:

```
p <nome> - imprime os telefones correspontes, ou "não existe!".
i <nome> <telefone> Insere um novo nome e telefone na lista (a não ser que esse par
já exista).
d <nome> Elimina da lista todas as ocorrências de pares (nome, telefone) com o nome
indicado.

Exemplo:
    p rui
        22666441
    i rui 888888
    p rui
        226664441
        888888
    d rui
    p rui
        não existe!
```

4. *Strings...*

Para a implementação das seguintes funções, consulte o manual da funções correspondentes da biblioteca *sting.h*.

- Implemente uma função `int my_strlen(char *s)`, que retorna o tamanho da string apontada por **s**.
- Implemente uma função `int my_strcmp(char *s1, char *s2)`, que compara a string apontada por **s1** à string apontada por **s2**. Deverá retornar um valor:
  - menor que 0, se a string **s1** for lexicograficamente menor que **s2**;
  - igual a 0, se a string **s1** for lexicograficamente igual a **s2**;
  - maior que 0, se a string **s1** for lexicograficamente maior que **s2**.
- Implemente uma função `int my_strncpy(char *src, char *dest, int n)`, que copia, no máximo, **n** caracteres da string **src**, para o destino **dest**, retornando um apontador para a string resultante **dest**. Que cuidados é necessário ter na implementação e utilização desta função?
- Implemente uma função `char* my_strdup(char *s)`, que retorna uma nova string corresponde a uma cópia da string **s**.
- Implemente uma função `char* my_strncat(char *dest, char *src)`, que acrescenta (copia), no máximo **n** caracteres da string **src** ao fim da string **dest**, retornando um apontador para a string resultante **dest**. Que cuidados é necessário ter na implementação e utilização desta função?