Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação em Sistemas Computacionais

Inverno de 2017/2018

Série de Exercícios 1 - Individual

Realize os exercícios seguintes usando a linguagem C. Não se esqueça de testar devidamente o código desenvolvido, bem como de o apresentar de forma cuidada, apropriadamente indentado e comentado. Assegure-se de que o compilador não emite qualquer aviso sobre o seu código, mesmo com a opção **`-Wall`** activa. Contacte o docente se tiver dúvidas. Não é necessário relatório. Encoraja-se a discussão de problemas e soluções com outros colegas, mas a partilha directa de soluções leva, no mínimo, à anulação das entregas de todos os envolvidos.

1. Escreva a função **`int_change`** que procura, no inteiro especificado com **`value`**, a primeira ocorrência de uma sequência com **`nbits`** *bits* especificada com o argumento **`pattern_to_change`** e retorna o resultado da substituição dessa sequência de *bits* em **`value`** pela sequência, com a mesma dimensão, especificada pelos **`nbits`** de menor peso do argumento **`new_pattern`**. Por exemplo, a chamada **`int_change(0x2AD555BC, 5, 0x15, 0x2A)`** deve retornar **`0x2AD54ABC`**.

   ```
   int int_change(int value, int nbits, int pattern_to_change, int new_pattern);
   ```

2. Escreva a função **`itoa`**, que não existe implementada na biblioteca *standard* da linguagem C, sem recorrer a outras funções dessa biblioteca. A função converte o inteiro **`value`** numa *string* com a sua representação em texto usando a base recebida no argumento **`base`** e guarda o resultado da conversão no *array* especificado pelo argumento **`str`**. Se a base for decimal (base 10) e **`value`** for negativo, a *string* resultante deverá incluir o sinal '-'. Nos restantes casos deverá considerar **`value`** no domínio dos números naturais. A função retorna o valor passado com o argumento **`str`**.

   ```
   char *itoa(int value, char *str, int base);
   ```

3. Escreva um programa de teste da função **`itoa`** que apresenta no *standard output* o resultado da conversão para cada valor introduzido no *standard input*. Os valores a converter são sempre definidos na base decimal e a base de conversão é definida por argumento do programa. Cada linha apresentada no *standard output* deve ser formada por: **`"nnn: len: vvv"`**, sendo **`nnn`** o número da linha na entrada, **`len`** o número de caracteres que formam o valor de acordo com a base definida e **`vvv`** a representação do valor convertido. Leia as linhas do *standard input* com a função *standard* **`fgets`**, usando **`stdin`** como terceiro argumento. O programa deve terminar quando ocorrer a primeira linha vazia no *standard input*. No programa de teste poderá recorrer a funções da biblioteca *standard* da linguagem C.

4. Escreva a função **`round_to_int`**, que arredonda o valor especificado pelo argumento **`fvalue`**, devolvendo o resultado do arredondamento através do parâmetro **`ivalue`**. O valor em vírgula flutuante (precisão simples) deverá ser convertido para o inteiro mais próximo, isto é, se a parte fracionária for maior ou igual a **`0,5`** o resultado da conversão é o valor inteiro acima; no caso contrário, será a parte inteira do número real. No caso de existir erro, isto é, o resultado não ser codificável numa variável do tipo **`int`** (*overflow*), a função devolve **`-1`**; no caso contrário, devolve **`0.`**

   Nota: Na implementação da função só podem ser utilizadas operações aritméticas e lógicas sobre inteiros. Qualquer operação de vírgula flutuante invalida o exercício.

   ```
   int round_to_int(float fvalue, int *ivalue);
   ```

5. Uma instância do tipo **BankingNode** contém informação sobre um movimento bancário. O campo **date** corresponde à data do movimento, o campo **description** contém a descrição do movimento, o campo **category** indica o tipo de despesa e o campo **next** permite a ligação de várias instâncias deste tipo em lista simplesmente ligada.

   A função **get_banking_by_category** recebe no argumento **movements** um *array* de instâncias do tipo **BankingNode** e agrupa-as em listas ligadas segundo a categoria da despesa (as instâncias não estão inicialmente ligadas em lista). As listas assim formadas são inscritas no *array* **categories**, em que cada índice corresponde directamente a uma categoria de despesa. Cada posição do *array* **categories** contém o ponteiro para o primeiro elemento da respetiva lista. Nas listas, os movimentos bancários devem ser ordenados por ordem crescente da data associada ao movimento.

   a) Defina adequadamente o tipo **Date** de modo a facilitar a comparação de datas.
   b) Implemente a função **get_banking_by_category.**
   c) Escreva um programa de teste que envolva o processamento de pelo menos nove movimentos pertencentes a três categorias.

```
typedef struct banking_node {
  Date date;
  char *description;
  int category;
  struct banking_node *next;
} BankingNode;

void get_banking_by_category(BankingNode *categories[], BankingNode movements[],
                             size_t num_movements);
```

Data limite de entrega: 22 de Outubro de 2017

ISEL, 29 de Setembro de 2017