Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação de Sistemas Computacionais

Inverno de 2020/2021

Série de Exercícios 1

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. Considere o critério "complemento para 2" na representação de valores inteiros relativos (com sinal). Escreva a função **`max_value`**, que retorna o maior valor positivo que pode ser armazenado pelo número de *bits* equivalente ao número de *bits* de um **`char`**, vezes o valor indicado pelo parâmetro **`nchars`**. Se o valor não for representável num **`unsigned long`**, deve retornar zero. Valorizam-se implementações que tenham em conta o número de *bits* por **`char`** indicado pela macro CHAR_BIT.

   ```
   unsigned long max_value(size_t nchars);
   ```

2. Considere vetores de *bits* representados sobre *arrays* de **`unsigned long`** em que **`ULONG_BIT`** é o número de *bits* de um **`unsigned long`**. No índice **`n`** de um *array* de **`unsigned long`** o *bit* de menor peso corresponde ao índice **`n * ULONG_BIT`** do vector de *bits*, enquanto que o *bit* de maior peso corresponde ao índice **`(n + 1) * ULONG_BIT - 1`**. A função **`getbits`** retorna o valor dos *bits* entre as posições **`idx`** e **`idx + len - 1`** do vetor de *bits* representado por **`data`**. A função **`setbits`** escreve os **`len`** *bits* de menor peso de **`val`** nas posições entre **`idx`** e **`idx + len - 1`** do vetor de *bits* representado por **`data`**. Em ambas as funções, **`len`** nunca é maior do que **`ULONG_BIT`**. Exemplo: para **`data = {0xBFFFFFECABCD1234, 0xC, 2, 3 }`**, a chamada a **`getbits(data, 29, 8)`** retorna **`0x0000000000000065`**. Defina **`ULONG_BIT`** e implemente as funções **`getbits`** e **`setbits`**.

   ```
   unsigned long getbits(unsigned long data[], size_t idx, size_t len);

   void setbits(unsigned long data[], size_t idx, size_t len, unsigned long val);
   ```

3. Programe em linguagem C a função **`string_split`**, que separa, em palavras isoladas, o texto recebido no parâmetro **`text`** com formato *string* C. Palavra é uma sequência de caracteres delimitada por sequências de um ou mais caracteres separadores (**`' '`**, **`'\t'`** ou **`'\n'`**). O array de ponteiros **`words`**, cuja dimensão é definida por **`words_size`**, deve ser preenchido com os endereços de início das palavras, pela ordem em que se encontram no texto. No caso de o número de palavras ser superior a **`words_size`**, preenche apenas as posições existentes. O caractere separador, no final de cada palavra, deve ser substituído por **`'\0'`**, para que as palavras fiquem formatadas como *strings* C. A função devolve o número de palavras encontradas.

   ```
   size_t string_split(char *text, char *words[], size_t words_size);
   ```

4. Programe, sem usar operações de **`float`**, a função **`print_float`**, que representa, na forma de *string* C, o valor real recebido no parâmetro **`value`**. A *string* deve ser depositada no *array* de caracteres apontado por **`buffer`**, cuja dimensão é indicada em **`buffer_size`**. A representação deve ser em notação decimal, com uma aproximação de seis casas decimais. A função devolve a dimensão da *string* formada.

   ```
   size_t print_float(char *buffer, size_t buffer_size, float value);
   ```

   Nota: na implementação interna da função não podem ser utilizadas operações aritméticas ou lógicas de **`float`** nem funções que operem sobre valores do tipo **`float`**. Qualquer operação de vírgula flutuante invalida a resolução do exercício.

5. Programe a função **`string_to_time`** que converte a informação de data e hora, recebida na forma de *string* C, para uma *struct* do tipo **`struct tm`**[1]. A *string* de entrada tem o formato "**`dd-mm-aaaa hh:mm:ss`**". Os campos **`tm_wday`**, **`tm_yday`** e **`tm_isdst`** devem ser colocados a zero.

---

[1] http://www.cplusplus.com/reference/ctime/tm/

```
struct tm {
  int   tm_sec;
  int   tm_min;
  int   tm_hour;
  int   tm_mday;
  int   tm_mon;
  int   tm_year;
  int   tm_wday;
  int   tm_yday;
  int   tm_isdst;
};

void string_time(char *string, struct tm *tm);
```

6. Realize um programa para operar sobre ficheiros de dados em formato CSV[2]. O programa deve realizar a operação definida em **<operação>** e produzir um novo ficheiro com o resultado.

   $ csv <operação> <ficheiro fonte>

   O campo **<operação>** pode assumir as seguintes formas e significados:

   -o <ficheiro destino> - especificar o nome do ficheiro com o resultado;

   -d <column> - apagar a coluna indicada por **<column>**;

   -f <column> <pattern> - selecionar as linhas que contenham o texto **<pattern>** na coluna indicada;

   -n normalizar - inserir separadores, criando células vazias, nas linhas menores que a maior linha existente;

   -s <antigo separador> <novo separador> - mudar o caractere usado como separador

   Exemplo:

   Considerando o ficheiro **stock.csv** com o seguinte conteúdo:

   ```
   máquina lavar roupa, Indesit, Lisboa, L3
   máquina lavar loiça, Bosch, Porto, D4
   torradeira, Moulinex, Lisboa, A10
   microondas, Samsung, Porto, E4
   ```

   o comando

   $ csv -f 2 Lisboa -o stock_lisboa.csv stock.csv

   produz o ficheiro **stock_lisboa.csv** com o seguinte conteúdo:

   ```
   máquina lavar roupa, Indesit, Lisboa, L3
   torradeira, Moulinex, Lisboa, A10
   ```

Data recomendada de entrega: 8 de Novembro de 2020

ISEL, 14 de Outubro de 2020

[2] https://pt.wikipedia.org/wiki/Comma-separated_values