Instituto Superior de Engenharia de Lisboa

LEETC

Programação II

2021/22 – 1.º semestre letivo

Primeira Série de Exercícios

1. Exploração de operações *bitwise* e deslocamentos

Pretende-se uma função para identificar a quantidade de bits a 1 na extremidade de maior peso de um valor com o tipo `char`. Deve ter em conta a portabilidade, usando o valor de `CHAR_BIT` (definido no *header file* `limits.h`) que indica a dimensão em *bits* de um `char`.

1.1. Escreva a função

```
int charLeadingOnes( char value );
```

que retorna o número de bits consecutivos com o valor 1, contados a partir do bit de maior peso do parâmetro `value`.

Por exemplo, se o parâmetro `value` tiver valores iniciados na parte alta pelas sequências de bits: 0, retorna 0; 10, retorna 1; 110, retorna 2; 1110, retorna 3; etc., até à dimensão total do tipo `char`.

1.2. Escreva um programa de teste e demonstração para a função anterior, ensaiando nomeadamente os valores no limite da representação e exemplos de casos em que alguns *bits* de menor peso sejam irrelevantes para o resultado.

2. Manipulação de *strings*

Pretende-se o processamento de *strings* para uniformizar a separação entre palavras, colocando um espaço apenas entre palavras, em substituição de qualquer sequência de separadores, espaço (‘ ’), tabulação (‘\t’) ou mudança de linha (‘\n’). Se houver separadores no início ou no final das *string*, devem ser eliminados.

2.1. Escreva a função

```
void separatorUnify( char str[] );
```

recebendo uma *string* no parâmetro `str`, aplica a formatação especificada, deixando a *string* processada no início do espaço anteriormente ocupado pela *string* original (note que a nova forma tem sempre um número de carateres igual ou inferior à original.) São valorizadas as soluções que não necessitem de memória temporária para o processamento. Pode realizar a modificação da *string* por manipulação direta ou usar os mecanismos declarados nos *header files* normalizados, nomeadamente `ctype.h` e `string.h`.

2.2. Escreva um programa de teste e demonstração para a função anterior. O programa recebe, através de *standard input*, um texto e reproduz, em *standard output*, o seu conteúdo após reformatado pela função `separatorUnify`. Propõe-se que faça o processamento linha a linha, usando a função `fgets`, de biblioteca, na leitura do texto de entrada.

Para o alojamento das *strings* propõe-se, por simplificação, que crie *arrays* com dimensão fixa, superior à da maior linha. Sugere-se que use o comando “`wc`” (*word count*), usando a opção “`-L`” para obter a dimensão da maior linha de um ficheiro.

O programa pode ser ensaiado com redireccionamento de *input* a partir de ficheiros de texto.

3. Estudo de *strings* codificadas em UTF-8

A norma UTF-8 para codificação de caracteres, adotada na plataforma Linux que utilizamos, permite a representação de caracteres compostos, por exemplo com acentuação ou cedilha. Trata-se de uma representação multi-*byte* com dimensão variável, de um a quatro *bytes*, destinada a representar de forma eficiente os códigos definidos na norma Unicode.

Em UTF-8, os caracteres compatíveis com a tabela ASCII básica, na gama de índices de 0 a 0x7f, são codificados com a dimensão de um *byte*. As sequências com mais *bytes* representam os carateres compostos.

Em cada sequência destas, o primeiro *byte* tem os dois ou mais *bits* de maior peso a 1, cuja quantidade representa o número total de *bytes* da sequência; os restantes *bytes* da sequência o têm o *bit* de maior peso a 1 e o adjacente a 0.

Recomenda-se a consulta complementar: https://en.wikipedia.org/wiki/UTF-8; http://www.utf8-chartable.de/; www.fileformat.info/info/unicode/utf8.htm.

3.1. Escreva uma função destinada a apresentar, através de *standard output*, uma parte do conteúdo de um *array* de `char`:

```
void printBytes( char a[], int i, int n );
```

A função deve mostrar `n` *bytes* existentes, a partir do índice `i`, no *array* `a`. Apresenta, entre chavetas, a sequência de valores, em hexadecimal a dois algarismos.
Por exemplo, se existir a definição "`char x[] = { 4, 1, 5, 0xff, 0x80 };`",
a chamada "`printBytes( x, 2, 3 );`" deve apresentar "`{05 ff 80}`".

3.2. Escreva uma função para identificar o número de *bytes* da representação de um símbolo em UTF-8:

```
int utf8Length( char a[], int i );
```

A função recebe, no parâmetro `a`, um *array* cujo elemento de índice `i` é o primeiro *byte* de um carater. Retorna o número total de *bytes* da sequência; pode ser: 1, no caso de ASCII básico, ou uma quantidade superior, até quatro, no caso de carateres compostos. Propõe-se que utilize a função do exercício 1.

3.3. Escreva um programa capaz de receber, através de *standard input*, um texto codificado em UTF-8, uniformizar o conteúdo das sua linhas, usando a função `separatorUnify`, e reproduzi-lo, em *standard output*, utilizando as funções `utf8Length` e `printBytes` para mostrar, após cada carater especial, a sequência de valores da sua codificação.
Por exemplo, se receber a frase

```
Permite usar acentuação     e outros símbolos      como «Euro» €.
```

deve apresentar

```
Permite usar acentuaç{c3 a7}ã{c3 a3}o e outros sí{c3 ad}mbolos como
«{c2 ab}Euro»{c2 bb} €{e2 82 ac}.
```

Propõe-se que faça o processamento linha a linha, usando na leitura a função `fgets` de biblioteca. Para o alojamento das *strings* sugere-se, por simplificação, que crie *arrays* com dimensão fixa, superior à da maior linha. Poderá definir arbitrariamente uma dimensão máxima de linha para os ficheiros a usar ou determiná-la em ficheiros existentes com o comando "`wc -L`", como proposto no exercício 2.

3.4. Teste o programa. Prepare um conjunto de ficheiros de texto para teste e demonstração, usados em redireccionamento de *standard input*; estes devem conter sequências de diversos grupos e várias dimensões na codificação UTF-8. Propõe-se que verifique a codificação de todos os carateres acentuados da língua portuguesa.

ISEL, 2021.10.20