INF1018 - Software Básico (2014.2)
Primeiro Trabalho

Leia com atenção o enunciado do trabalho e as instruções para a entrega. Em caso de dúvidas, não invente. Pergunte!

Codificação UNICODE
Em computação, caracteres de texto são tipicamente representados por códigos especificados por algum padrão de codificação. Um padrão bastante conhecido é a codificação ASCII, que utiliza valores inteiros de 0 a 127 para representar letras, dígitos e alguns outros simbolos. Algumas extensões dessa codificação utilizam também a faixa de valores de 128 a 255 para representar, por exemplo, caracteres acentuados e alguns outros símbolos adicionais.

A codificação ASCII e outros padrões de codificação que utilizam códigos de um único byte são limitados à representação de apenas 256 símbolos diferentes. Para permitir a representação de um conjunto maior de caracteres foi criada, no final da década de 1980, a codificação UNICODE. A versão corrente dessa codificação é capaz de representar os caracteres utilizados por todos os idiomas conhecidos, além de diversos outros símbolos.

Cada caractere em UNICODE é associado a um código na faixa de 0 a 0x10FFFF, o que permite a representação de 1.114.112 símbolos diferentes. Na notação adotada pelo padrão UNICODE, U +xxxx identifica o código com valor hexadecimal xxxx. Por exemplo, o código U+00A9 (o código do símbolo ©) corresponde ao valor hexadecimal 0x00A9.

Existem algumas formas diferentes de codificação de caracteres UNICODE. Para este trabalho, as codificações de interesse são a UTF-8 e a UTF-32.

Codificação UTF-32

A codificação UTF-32 é simplesmente a representação numérica do código do caracter, em um inteiro de 4 bytes. Essa forma de codificação é suficiente para representar todos os códigos previstos no UNICODE, não sendo necessária maior elaboração.

Codificação UTF-8

Na codificação UTF-8, os códigos dos caracteres são representados em um número variável de bytes. O tamanho mínimo utilizado para representar um caractere em UTF-8 é um byte (8 bits); se a representação necessita de mais espaço, mais bytes são utilizados (até o máximo de 4 bytes). Uma característica importante é que a codificação UTF-8 é compatível com o padrão ASCII, ou seja, os 128 caracteres associados aos códigos de 0 a 0x7F em ASCII tem a mesma representação (em um único byte) em UTF-8.

A tabela a seguir indica para cada faixa de valores de códigos UNICODE o número de bytes necessários para representá-los e a codificação usada para essa faixa.

Código UNICODE	Representação UTF-8 (byte a byte)
U+0000 a U+007F	0xxxxxxx
U+0080 a U+07FF	110xxxxx 10xxxxxx
U+0800 a U+FFFF	1110xxxx 10xxxxxx 10xxxxxx
U+10000 a U+10FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Note que:

x é o valor de um bit. O bit x da extrema direita é o bit menos significativo.
Apenas a menor representação possível de um caractere deve ser utilizada.
Um código representado em um único byte tem sempre 0 no bit mais significativo.

Se um código é representado por uma sequência de bytes, o número de 1s no início do primeiro byte indica o número total de bytes da sequência. Esses 1s são seguidos sempre por um 0. Todos os bytes que se seguem ao primeiro começam com 10 (indicando que é um byte de continuação).
Exemplos:
O símbolo © tem código UNICODE U+00A9.
Em binário A9 é 1010 1001. Usando a codificação de 2 bytes para a faixa U+0080 a U+07FF temos:
11000010 10101001 = 0xC2 0xA9

Note o preenchimento com zeros à esquerda para completar a porção do código do caractere colocada no primeiro byte da sequência.

O símbolo ≠ tem código UNICODE U+2260.
Em binário 2260 é 0010 0010 0110 0000. Usando a codificação de 3 bytes para a faixa U+0800 a U+FFFF temos:
11100010 10001001 10100000 = 0xE2 0x89 0xA0

Byte Order Mark (BOM)

O caractere BOM (código U+FEFF) é um caractere especial opcionalmente inserido no início de um arquivo que contenha texto codificado em UNICODE.
O BOM funciona como uma assinatura do arquivo: além de identificar o conteúdo do arquivo como UNICODE ele também define a ordem de armazenamento do arquivo (big-endian ou little-endian). Como a ordem de armazenamento não faz sentido para arquivos UTF-8, o BOM é raramente usado no início de arquivos UTF-8, e muitas aplicações que trabalham com UTF-8 não dão suporte a um BOM no inicio de um arquivo de entrada.

A sequência exata de bytes que corresponde ao caractere BOM inserido no início do arquivo depende da codificação empregada e da ordem de armazenamento do arquivo, como mostrado a seguir:

Tipo do Arquivo	Bytes
UTF-32, big-endian	00 00 FE FF
UTF-32, little-endian	FF FE 00 00

Objetivo do trabalho
O objetivo do trabalho é implementar, na linguagem C, duas funções (conv8_32 e conv32_8), que recebem como entrada um arquivo contendo um texto codificado em um dos dois formatos (UTF-8 ou UTF-32) e geram como saida um arquivo contendo o mesmo texto, codificado no outro formato.

Conversão UTF-8 para UTF-32:

A função conv8_32 deve ler o conteúdo de um arquivo de entrada (um texto codificado em UTF-8) e gravar em um arquivo de saída esse mesmo texto, codificado em UTF-32.
O arquivo de entrada UTF-8 não terá um caractere BOM inicial. Já o arquivo de saída deverá necessariamente conter o BOM inicial.

O protótipo (cabeçalho) da função conv8_32 é o seguinte:

```
int conv8_32(FILE *arq_entrada, FILE *arq_saida, char ordem);
```

Os dois primeiros parâmetros da função são dois arquivos binários já abertos: o arquivo de entrada (arq_entrada) e o arquivo de saída (arq_saida). O terceiro parâmetro especifica a ordenação do arquivo de saída:

'L' indica ordenação little-endian
'B' indica ordenação big-endian
O valor de retorno da função conv8_32 é 0, em caso de sucesso e -1, em caso de erro. Dois tipos de erros podem ocorrer:
erro de E/S: neste caso, a função deve emitir, na saída de erro (stderr), uma mensagem indicando o tipo de erro ocorrido (leitura ou gravação)
codificação incorreta no arquivo de entrada: neste caso, a função deve emitir, na saída de erro (stderr) uma mensagem indicando a posição no arquivo de entrada onde o erro foi detetado (o índice do primeiro byte inconsistente, considerando-se que o primeiro byte do arquivo é o byte 0).
Nos dois casos, após emitir a mensagem de erro, a função deve retornar.
Conversão UTF-32 para UTF-8:

A função conv32_8 deve ler o conteúdo de um arquivo de entrada (um texto codificado em UTF-32) e gravar em um arquivo de saída esse mesmo texto, codificado em UTF-8. O protótipo da função é o seguinte:

```
int conv32_8(FILE *arq_entrada, FILE *arq_saida);
```

Os parâmetros da função são dois arquivos binários abertos: o arquivo de entrada (arq_entrada) e o arquivo de saída (arq_saida).
O arquivo de entrada deverá necessariamente conter o BOM inicial. O arquivo de saída UTF-8 não deverá conter um caractere BOM inicial.

Note que, neste caso, a própria função descobre a ordenação do arquivo de entrada (UTF-32), inspecionando o BOM no início do arquivo.

Assim como na função anterior, o valor de retorno é 0, em caso de sucesso e -1, em caso de erro. Da mesma forma, os procedimentos para os casos de erro são:

erro de E/S: a função deve emitir, na saída de erro (stderr), uma mensagem indicando o tipo de erro ocorrido (leitura ou gravação)
codificação incorreta no arquivo de entrada: a função deve emitir, na saída de erro (stderr) uma mensagem indicando a posição no arquivo de entrada onde o erro foi detetado (o índice do primeiro byte inconsistente, considerando-se que o primeiro byte do arquivo é o byte 0).
Implementação e Execução
Você deve criar um arquivo fonte chamado utfconv.c contendo as duas funções descritas acima, e funções auxiliares, se for o caso. Esse arquivo não deve conter uma função main!

Crie também um arquivo utfconv.h , que deve conter apenas os protótipos das funções conv32_8 e conv8_32. Mantenha os protótipos especificados no enunciado do trabalho, sem quaisquer alterações!

Para testar seu programa, crie um outro arquivo, por exemplo, teste.c, contendo uma função main. Crie seu programa executável, por exemplo teste, com a linha:

```
gcc -Wall -o teste utfconv.c teste.c
```

Tanto o arquivo utfconv.c como teste.c deve conter a linha:

```
#include "utfconv.h"
```

Exemplos para teste
Para testar suas funções você poderá usar os arquivos a seguir:

Arquivos sem erros

Arquivos pequenos, sem erros: os três arquivos contém o mesmo "texto", codificado em UTF-8 e UTF-32 (little-endian e big-endian)
Arquivo Pequeno UTF-8

Arquivo Pequeno UTF-32 (little endian)
Arquivo Pequeno UTF-32 (big endian)
Arquivos grandes, sem erros: os três arquivos contém o mesmo "texto", codificado em UTF-8 e UTF-32 (little-endian e big-endian)
Arquivo Grande UTF-8
Arquivo Grande UTF-32 (little endian)
Arquivo Grande UTF-32 (big endian)
Arquivos com erros

Arquivos utf-8: utf8_inv_1, utf8_inv_2, utf8_inv_3
Arquivos utf-32: utf32_inv_1, utf32_inv_2, utf32_inv_3, utf32_inv_4, utf32_inv_5

Entrega

Devem ser entregues via Moodle dois arquivos:

o arquivo utfconv.c
Coloque no início do arquivo fonte, como comentário, os nomes dos integrantes do grupo, da seguinte forma:

```
  /* Nome_do_Aluno1 Matricula Turma */
  /* Nome_do_Aluno2 Matricula Turma */
```

um arquivo texto, chamado relatorio.txt, explicando o que está funcionando e, eventualmente, o que não está funcionando. Se o seu programa tem bugs, você saber disso é um atenuante...
Coloque também no relatório o nome dos integrantes do grupo.
Coloque também na área de texto da tarefa o nome dos integrantes do grupo

Apenas uma entrega é necessária (usando o login de um dos integrantes do grupo)

Prazo

O trabalho deve ser entregue até a meia-noite do dia 26 de setembro
Trabalhos entregues com atraso perderão um ponto por dia de atraso.

Observações

Os trabalhos devem preferencialmente ser feitos em grupos de dois alunos .
Alguns grupos poderão ser chamados para apresentações orais / demonstrações dos trabalhos entregues.