INF1018 - Software Básico (2022.2)
Primeiro Trabalho

Conversão entre codificações UNICODE

O objetivo deste trabalho é implementar, na linguagem C, duas funções (**converteUtf8Para32** e **converteUtf32Para8**), que recebem como entrada um arquivo contendo um texto codificado em um formato UNICODE (UTF-8 ou UTF-32) e geram como saída um arquivo contendo o mesmo texto, codificado no outro formato.

Instruções Gerais

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

- O trabalho deve ser entregue **até meia-noite (23:59) do dia 09 de outubro**.
- Trabalhos entregues com atraso perderão **um ponto por dia de atraso**.
- Trabalhos que não compilem (isto é, que não produzam um executável) **não serão considerados!** Ou seja, receberão grau zero.

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

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 símbolos. 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-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 **1**s 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:

1. 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:

   11**000010** 10**101001** = 0xC2 0xA9

   O primeiro byte começa com 110, indicando que a sequência é composta por dois bytes. A seguir vêm os cinco primeiros bits do código UNICODE (note o preenchimento com zeros à esquerda para completar a porção do código do caractere colocada no primeiro byte da sequência).

   O segundo byte começa com 10, indicando que é um byte de continuação. A seguir vêm os próximos seis bits do código UNICODE.

2. 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:

   1110**0010** 10**001001** 10**100000** = 0xE2 0x89 0xA0

   O primeiro byte começa com 1110, indicando que a sequência é composta por três bytes. A seguir vêm os quatro primeiros bits do código UNICODE

   O segundo e o terceiro bytes começam com 10, indicando que são bytes de continuação. Cada um deles tem, em seguida, os próximos seis bits do código UNICODE.

Codificação UTF-32

Na codificação UTF-32, os códigos dos caracteres são representados em um inteiro de 32 bits.

Exemplos:

- O caractere 'z minúsculo' tem código UNICODE **U+007A** e é representado em UTF-32 por **0x0000007A**.
- O caractere grego β (beta minúsculo) tem código UNICODE **U+03B2** e é representado em UTF-32 por **0x000003B2**.
- O símbolo Han 水 (água) tem código UNICODE **U+6C34** e é representado em UTF-32 por **0x00006C34**.
- O símbolo 𝄞 (clave de sol) tem código UNICODE **U+1D11E** e é representado em UTF-32 por **0x0001D11E**.

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 seu uso não é recomendado pelo padrão. Além disso, muitas aplicações que trabalham com UTF-8 não dão suporte a um BOM no início do arquivo.

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. Para a codificação UTF-32, temos:

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

Funções de Conversão

O objetivo deste trabalho é implementar, na linguagem C, as funções **converteUtf8Para32** e **converteUtf32Para8**, que realizam a conversão de um formato UNICODE (UTF-8 ou UTF-32) para o outro formato.

Conversão UTF-8 para UTF-32

A função **converteUtf8Para32** 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, **com ordenação LITTLE-ENDIAN** .

O arquivo de entrada UTF-8 não terá um caracter BOM inicial; já o arquivo de saída, UTF-32, deverá necessariamente conter o BOM inicial.

O protótipo (cabeçalho) da função **converteUtf8Para32** é o seguinte:

```
int converteUtf8Para32(FILE *arquivo_entrada, FILE *arquivo_saida);
```

Os dois parâmetros da função são dois arquivos abertos em modo binário: o arquivo de entrada (arquivo_entrada) e o arquivo de saída (arquivo_saida).

O valor de retorno da função **converteUtf8Para32** é 0, em caso de sucesso e -1, em caso de erro de E/S. Em caso de erro, a função deve emitir, na saída de erro (stderr), uma mensagem indicando qual o tipo de erro ocorrido (leitura ou gravação) e retornar imediatamente.

Por simplicidade, você pode considerar que os arquivos de entrada sempre conterão um texto CORRETAMENTE CODIFICADO. Dessa forma, você não precisa implementar o tratamento de erros de codificação do arquivo de entrada.

Conversão UTF-32 para UTF-8

A função **converteUtf32Para8** deve ler o conteúdo de um arquivo de entrada (um texto codificado em UTF-32 **com ordenação indicada pelo BOM)** e gravar em um arquivo de saída esse mesmo texto, codificado em UTF-8.

O arquivo de entrada UTF-32 terá um BOM inicial, mas o arquivo de saída UTF-8 não deverá conter um BOM inicial.

O protótipo da função é o seguinte:

```
int converteUtf32Para8(FILE *arquivo_entrada, FILE *arquivo_saida);
```

Os parâmetros da função são dois arquivos abertos em modo binário: o arquivo de entrada (arquivo_entrada) e o arquivo de saída (arquivo_saida).

A função deve inspecionar os primeiros quatro bytes do conteúdo do arquivo de entrada (UTF-32) para verificar se eles contêm um BOM válido!

Assim como na função anterior, o valor de retorno é 0, em caso de sucesso e -1, em caso de erro. 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)
- BOM inválido: a função deve emitir, na saída de erro (stderr) uma mensagem de erro.

Nos dois casos, a função deve retornar imediatamente após emitir a mensagem de erro.

Por simplicidade, você pode considerar que os arquivos de entrada sempre conterão um texto CORRETAMENTE CODIFICADO. Apenas o caso de BOM inválido (ou ausente) no início do arquivo precisa ser tratado.

Implementação e Execução

Você deve criar um arquivo fonte chamado **converteutf.c** contendo as duas funções descritas acima, e funções auxiliares, se for o caso. Esse arquivo **não** deve conter uma função main!

O arquivo converteutf.c deverá incluir o arquivo de cabeçalho **converteutf.h** , fornecido aqui.

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

```
gcc -Wall -o testeconv converteutf.c testeconv.c
```

Tanto o arquivo **converteutf.c** como **testeconv.c** devem conter a linha:

```
#include "converteutf.h"
```

Exemplos para teste

Para testar suas funções você pode usar os arquivos fornecidos a seguir:

- Arquivos pequenos: os dois arquivos contêm o mesmo "texto", codificado em UTF-8 e UTF-32
  - Arquivo Pequeno UTF-8
  - Arquivo Pequeno UTF-32 (little_endian)

Dicas

Implemente seu trabalho por partes, testando cada parte implementada antes de prosseguir.

Por exemplo, você pode implementar primeiro a leitura de um arquivo UTF-8, decodificando os caracteres para códigos de 32 bits e exibindo esses valores na tela. Quando essa parte estiver funcionando, implemente a codificação UTF-32 sobre os valores, gerando um arquivo de saída.

Para verificar o conteúdo do arquivo gravado, você pode usar o utilitário **hexdump**. Por exemplo, o comando

```
hexdump -C <nome-do-arquivo>
```

exibe o conteúdo do arquivo especificado byte a byte, em hexadecimal (16 bytes por linha). A segunda coluna de cada linha (entre '|') exibe os caracteres ASCII correspondentes a esses bytes, **se eles existirem**. Experimente inspecionar os arquivos dados como exemplo com o hexdump...

Para abrir um arquivo para gravação ou leitura em formato binário, use a função

```
FILE *fopen(char *path, char *mode);
```

descrita em stdio.h. Seus argumentos são:

- path: nome do arquivo a ser aberto
- mode: uma string que, no nosso caso, será **"rb"** para abrir o arquivo para leitura em modo binário ou **"wb"** para abrir o arquivo para escrita em modo binário.

A letra 'b', que indica o modo binário, é ignorada em sistemas como Linux, que tratam da mesma forma arquivos de tipos texto e binário. Mas ela é necessária em outros sistemas, como Windows, que tratam de forma diferente arquivos de tipos texto e binário (interpretando/modificando, por exemplo, bytes de arquivos "texto" que correspondem a caracteres de controle).

Para fazer a leitura e gravação do arquivo, uma sugestão é pesquisar as funções fwrite/fread e fputc/fgetc.

Entrega

Deverão ser entregues **via Moodle/EAD** dois arquivos:

1. Um arquivo fonte chamado **converteutf.c** , contendo as funções **converteUtf8Para32** e **ConverteUtf32Para8** (e funções auxiliares, se for o caso).
   - Esse arquivo **não** deve conter a função main, e só deve incluir o arquivo de cabeçalho converteutf.h e arquivos de cabeçalho da biblioteca padrão de C.
   - Coloque no início do arquivo, como comentário, os nomes dos integrantes do grupo da seguinte forma:

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

2. Um arquivo texto, chamado **relatorio.txt**, contendo um pequeno relatório.
   - O relatório deverá explicar o que está funcionando e o que não está funcionando. **Não é necessário documentar suas funções no relatório.** Seu código deverá ser claro o suficiente para que isso não seja necessário.
   - Coloque também no relatório o nome dos integrantes do grupo

Indique na área de texto da tarefa do Moodle o nome dos integrantes do grupo. Apenas uma entrega deve ser feita **se os dois integrantes pertencerem à mesma turma**.