INF1018 - Software Básico (2015.1)
Primeiro Trabalho

Gravação compactada

O objetivo do trabalho é implementar, na linguagem C, uma função (**`compacta`**) que escreve um array de structs em um arquivo binário de forma compacta (isto é, sem *padding*) e uma função (**`mostra`**) que permite visualizar um arquivo gerado por **`compacta`**.

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

Função `compacta`

```
int compacta (int nstructs, void* valores, char* campos, char ord, FILE* arquivo);
```

A função **`compacta`** recebe como argumentos:

- `nstructs`: o número de elementos do array de structs a ser escrito em arquivo
- `valores`: um ponteiro para o array de structs propriamente dito
- `campos`: uma descrição dos campos das structs que compõem o array
- `ord`: um caractere indicando se os valores básicos contidos nesses campos devem ser armazenados no arquivo em *little endian* ('L')ou *big endian* ('B')
- `arquivo`: um arquivo aberto para escrita, em modo binário

A função deverá retornar 0 em caso de sucesso, e -1 em caso de erro. Apenas erros de E/S (ou seja, erros na gravação do arquivo) devem ser considerados. Assuma que todos os parâmetros fornecidos à função estão corretos. A função `compacta` não deve fechar o arquivo de saída. Isso deverá ser feito pela função que abriu o arquivo (provavelmente, a `main`).

A string **`campos`** representa, na ordem, o tipo de cada campo das structs, de acordo com o código a seguir:

```
    'c' - char
    's' - short int
    'i' - int
```

Como exemplo, dada a declaração:

```
  struct s {
    int i1;
    short s1,s2;
    char c1;
    int i2;
  };
struct s exemplo[10];
```

a string `campos` correspondente é **`"issci"`**.

Assumindo que o descritor do arquivo de saída está armazenado em uma variável **`arq`**, do tipo **`FILE*`**, a chamada para a gravação compactada do array **`exemplo`** com ordenaçao *little-endian* seria:

```
res = compacta(10, exemplo, "issci", 'L', arq);
```

Formato do arquivo gerado

Os primeiros bytes do arquivo formam o cabeçalho, com informações sobre os dados armazenados. A seguir vem uma sequência de bytes contendo os dados propriamente ditos.

O formato do arquivo de saída deve ser o seguinte:

- o primeiro byte do cabeçalho indica o número de structs armazenadas no arquivo, como um `unsigned char`. Note que o número máximo de structs armazenada do arquivo é, portanto, 255.
- o bit mais significativo do segundo byte indica se os dados estão em *little_endian* (1) ou em *big_endian* (0).
- os próximos sete bits desse segundo byte indicam o número de campos de cada struct armazenada. Dessa forma, o número máximo de campos de cada struct é 127.
- a seguir aparecem os descritores de campos dos structs, cada um com dois bits. Cada descritor é codificado da seguinte forma:

  ```
  01 - char
  10 - short int
  11 - int
  ```

  A porção não utilizada do último byte de cabeçalho (se houver) deve ser prenchida com zeros (ou seja, o inicio dos dados propriamente ditos deve estar alinhado no próximo byte do arquivo).

Após o cabeçalho são armazenados os bytes com os dados do array de structs, na ordenação especificada. Não devem ser escritos no arquivo os bytes relativos a *padding*!

Voltando ao caso do array `exemplo`, os bytes no início do arquivo seriam:

```
| 0000 1010 | /* há 10 structs neste arquivo */
| 1000 0101 | /* ordenação little-endian, cada struct tem 5 campos */
| 1110 1001 | /* descrição dos primeiros quatro campos (i s s c)*/
| 1100 0000 | /* descrição do último campo (i) e preenchimento */
| xxxx xxxx | /* aqui começam os dados das 10 structs */
```

Nesse exemplo, na memória de um IA-32 executando Linux, cada struct ocuparia 16 bytes. Ao ser armazenada em arquivo por `compacta`, essa mesma struct ocuparia 13 bytes.

Função `mostra`

```
void mostra (FILE *arquivo);
```

A função `mostra` permite a visualização, na saída padrão, de um arquivo criado por `compacta`. Essa saída pode ser gerada, por exemplo, através de chamadas a `printf`.

O único argumento de `mostra` é o descritor de um arquivo aberto para leitura, em modo binário. Não é necessário considerar erros na leitura desse arquivo. A função `mostra` não deve fechar o arquivo de leitura. Isso deverá ser feito pela função que abriu o arquivo (provavelmente, a `main`).

A saída da função `mostra` deve ser a seguinte:

- uma linha indicando a ordenação do arquivo ("little" ou "big")
- uma linha indicando o número de structs armazenadas no arquivo (em formato decimal)
- "dump" dos valores armazenados, em hexa, com um campo por linha. Cada byte deve ser exibido em hexa, com dois dígitos. Deve haver um espaço entre cada dois bytes.
- uma linha separadora no início de cada struct (12 caracteres '-')

Como exemplo, para o mesmo arquivo discutido acima, a saída de `mostra` seria

```
little
10
------------
xx xx xx xx
xx xx
xx xx
xx
xx xx xx xx
------------
xx xx xx xx
xx xx
xx xx
xx
xx xx xx xx
------------
xx xx xx xx
xx xx
xx xx
xx
xx xx xx xx
------------
.... (etc, até completar as 10 estruturas)
```

onde os "xx" correspondem aos valores (em hexa) dos bytes armazenados.

Implementação e Execução

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

Crie também um arquivo **`compacta.h`** , que deve conter apenas os protótipos (cabeçalhos) das funções `compacta` e `mostra`.

Para testar seu programa, crie um outro arquivo, por exemplo, **`teste.c`** , contendo a função `main`.

Note que é responsabilidade da função `main` abrir o arquivo a ser gravado (por `compacta`) ou lido (por `mostra`). O descritor do arquivo aberto será passado, como parâmetro, para essas funções.

Crie seu programa executável, `teste`, com a linha:

```
gcc -Wall -m32 -o teste compacta.c teste.c
```

Tanto o arquivo **`compacta.c`** como **`teste.c`** devem conter a linha:

```
#include "compacta.h"
```

Dicas

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

Por exemplo, você pode implementar primeiro a gravação do arquivo compactado. Comece implementando casos simples (estruturas com campos do tipo 'char'), e vá introduzindo mais tipos de campos à medida que os casos anteriores estejam funcionando. Experimente diferentes tipos de alinhamento. Teste as diferentes ordenações (*little* e *big*).

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.

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

Devem ser entregues **via Moodle** dois arquivos:

1. o arquivo fonte **compacta.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 */
   ```

   Lembre-se que este arquivo não deve conter a função `main`!

2. um arquivo texto, chamado **relatorio.txt**, descrevendo os testes realizados, o que está funcionando e, eventualmente, o que não está funcionando. Mostre exemplos de estruturas testadas (casos de sucesso e insucesso, se houver)! Não é necessário explicar a sua implementação neste relatório. Seu programa deve ser suficientemente claro e bem comentado.

   Coloque também no relatório o nome dos integrantes do grupo.

Coloque na área de texto da tarefa do Moodle os nomes e turmas dos integrantes do grupo.

Para grupos de alunos da mesma turma, 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 04/05**
- 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.