INF1018 - Software Básico (2018.2) Primeiro Trabalho

Armazenando Estruturas de Inteiros

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

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 o final do dia 12/10 (23:59).
- Trabalhos entregues após o prazo perderão um ponto por dia de atraso.
- Trabalhos que não compilem (i.e., que não produzam um executável) não serão considerados, ou seja, receberão grau zero.
- Os trabalhos podem ser feitos em grupos de **no máximo** dois alunos.
- Alguns grupos poderão ser chamados para apresentar seus trabalhos.

Função `grava_structs`

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

A função **`grava_structs`** recebe:

- `nstructs`: o número de elementos do array de structs a ser escrito em arquivo
- `valores`: endereço do array de structs propriamente dito
- `campos`: descrição dos campos das structs que compõem o array
- `ord`: um caractere que indica se os valores inteiros contidos nas estruturas devem ser armazenados no arquivo em ordenação *little endian* ('L')ou *big endian* ('B')
- `arquivo`: o nome do arquivo a ser criado

A função deverá retornar 0 em caso de sucesso, e -1 em caso de erro. Apenas erros de E/S (erro na abertura ou gravação do arquivo) devem ser considerados. Assuma que todos os argumentos fornecidos à função estão corretos.

A string **`campos`** representa, na ordem dada, o tipo de cada campo das structs, conforme abaixo:

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

Por exemplo, dada a declaração:

```
struct s {
    int i1;
```

```
    long l1;
    short s1;
    char c1;
    short s2;
  };
struct s exemplo[10];
```

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

Assumindo que o nome do arquivo de saída é **saida1**, a chamada para a gravação do array de estruturas **`exemplo`** com ordenaçao *little-endian* seria:

```
    res = grava_structs(10, exemplo, "ilscs", 'L', "saida1");
```

Atenção! Para acessar os valores dos campos das estruturas (armazenados na memória), a função deve levar em consideração as regras de alinhamento especificadas para o ambiente onde ela será executada (SO Linux, em uma máquina de 64 bits).

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, portanto, o número máximo de structs armazenadas do arquivo é 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 é 127.
- a seguir aparecem os descritores de campos dos structs, cada um com dois bits. Cada descritor é codificado da seguinte forma:

  ```
  00 - char
  01 - short int
  10 - int
  11 - long 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 */
| 1011 0100 | /* descrição dos primeiros quatro campos (i l s c)*/
| 0100 0000 | /* descrição do último campos (s) e preenchimento com zeros */
| xxxx xxxx | /* aqui começam os dados das 10 structs */
```

Nesse exemplo, na memória de um sistema de 64 bits executando Linux, cada struct ocuparia 24 bytes. Ao ser armazenada em arquivo por `grava_structs`, essa mesma struct (sem *paddings*) ocuparia 17 bytes.

Função `dump_structs`

```
void dump_structs (char *arquivo);
```

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

O único argumento de `dump_structs` é o nome do arquivo. Em caso de erro na abertura ou leitura do arquivo, a função deverá emitir uma mensagem e retornar.

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

- uma linha indicando a ordenação do arquivo ("L" ou "B")
- 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 único espaço entre cada dois bytes.
- uma linha separadora no início de cada struct (caractere '*')

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

```
L
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 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 **`grava_structs.c`** contendo as duas funções descritas acima (`grava_structs e dump_structs`) e funções auxiliares, se for o caso. Esse arquivo **não deve conter uma função `main`!**

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

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

Você pode criar seu programa executável, `teste`, com a linha:

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

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 (que exijam/não exijam *paddings*). 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 **grava_structs.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 com alunos da mesma turma, apenas uma entrega é necessária (usando o *login* de um dos integrantes do grupo).

Se o grupo for composto por alunos de turmas diferentes, os dois alunos deverão realizar a entrega e avisar aos professores sobre a composição do grupo.