INF1018 - Software Básico (2022.1)
Primeiro Trabalho

Armazenamento compactado

O objetivo do trabalho é implementar, na linguagem C, uma função (**`gravacomp`**) que armazena um array de structs em um arquivo binário de forma compactada e uma função (**`mostracomp`**) que permite visualizar um arquivo gerado por **`gravacomp`** na saída padrão (terminal).

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 maio**.
- 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 (no máximo) dois alunos** .
- Os grupos deverão estar preparados para apresentações orais / demonstrações dos trabalhos entregues.

Função `gravacomp`

```
int gravacomp (int nstructs, void* valores, char* descritor, FILE* arquivo);
```

A função **`gravacomp`** recebe:

- `nstructs`: o número de elementos do array de structs a ser armazenado em arquivo
- `valores`: um ponteiro para o array de structs propriamente dito
- `descritor`: uma descrição dos campos das structs que compõem o array
- `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.

Atenção: a função não deve fechar o arquivo de saída! Isso deverá ser feito pela função que abriu o arquivo (provavelmente, a `main`).

A string **`descritor`** representa o tipo de cada um dos campos das structs (na ordem correeta), conforme abaixo:

```
   's' - string (char [])
   'u' - inteiro sem sinal (unsigned int)
   'i' - inteiro com sinal (signed int)
```

Para campos do tipo *string*, deve-se indicar o tamanho do array, **sempre com dois dígitos**. O tamanho máximo das *strings* armazenadas nas estruturas, por restrição imposta pelo formato do arquivo (como veremos mais adiante), é 63, descontado o marcador de fim (`\0`). Portanto, o tamanho máximo do array é limitado a 64.

Por exemplo, dada a declaração:

```
  struct s {
    int  i1, i2;
    char s1[3];
    unsigned int u1;
    char s2[10];
  };
struct s exemplo[5];
```

a string `descritor` correspondente é **`"iis03us10"`**.

Assumindo que o arquivo de saída está armazenado em uma variável **`arq`**, do tipo **`FILE*`**, a chamada para a gravação compactada do array **`exemplo`** (após preenchido) seria:

```
  res = gravacomp(5, exemplo, "iis03us10", arq);
```

Atenção! Para acessar os valores dos campos das estruturas (armazenados na memória), a função `code` deve levar em consideração as regras de alinhamento especificadas para o ambiente onde ela será executada (SO Linux, em uma máquina de 64 bits). **Os bytes de padding não devem ser armazenados no arquivo!**

Formato do arquivo gerado

O formato do arquivo de saída deve ser o seguinte:

- O primeiro byte do arquivo indica o número de structs armazenadas (como um *unsigned char*). Note que o número máximo de structs armazenadas no arquivo é, portanto, 255.
- A seguir deverão vir os campos de cada estrutura.

  Para cada campo, deverão ser gravados um *byte de cabeçalho* e, em seguida, os bytes que representam o conteúdo compactado do campo, conforme descrito a seguir.

Armazenamento de strings

Para campos do tipo *string*, o cabeçalho tem o seguinte formato:

BITS:	7	6	5-0
	cont	**tipo**	**tamanho**

O bit mais significativo (**cont**) indica se este é o último campo da estrutura (1) ou não (0). O bit 6 (**tipo**) deverá conter o valor 1. Os bits 5-0 (**tamanho**) têm o tamanho (em número de bytes) da string armazenada a seguir.

Após o cabeçalho, devem ser gravados os bytes que compõem a string, **porém apenas os bytes anteriores ao marcador de fim de string (\0)**. Note que o tamanho da string armazenada pode variar de 1 a n-1, onde n é o tamanho do vetor que corresponde a esse campo na estrutura (pois o \0 nunca será gravado). Como temos apenas 6 bits para o tamanho da string, esse tamanho é limitado a 63.

Por exemplo, se temos um campo definido como **`char s2[10]`**, que não é o último campo da estrutura, e neste campo está armazenada a string **`"abc"`**, seu cabeçalho terá:

- cont: 0
- tipo: 1
- tamanho: 3

e apenas três bytes serão gravados após o cabeçalho (os códigos dos caracteres 'a', 'b' e 'c').

Armazenamento de inteiros com e sem sinal

Para campos do tipo *inteiro*, o cabeçalho tem o seguinte formato:

BITS:	7	6-5	4-0
	cont	**tipo**	**tamanho**

Novamente, o bit mais significativo (**cont**) indica se este é o último campo da estrutura (1) ou não (0). Os bits 4-0 (**tamanho**) têm o número de bytes usados para representar o valor e os bit 6-5 (**tipo**) deverão conter

- 00 - se o campo é do tipo **u** (unsigned int)
- 01 - se o campo é do tipo **i** (signed int)

Após o cabeçalho, devem ser gravados, **em big endian**, os bytes que representam o valor do campo. Porém, ao invés de armazenar sempre quatro bytes, a função deverá gravar **apenas os bytes necessários para manter a representação do valor**.

Por exemplo, se temos um campo do tipo *signed int* com o valor **-1**, podemos usar apenas um byte para seu valor (um byte com valor **11111111** em binário, ou **FF** em hexadecimal). Desta forma, se esse campo é o último campo da estrutura, seu cabeçalho terá:

- cont: 1
- tipo: 01
- tamanho: 1

e após o cabeçalho será gravado apenas um byte

Se o campo contiver o valor **258**, precisaremos de dois bytes para representar seu valor (o primeiro byte com valor **01** e o segundo com valor **02**, em hexadecimal).

Se o campo contiver o valor -65536 (-2^16), precisaremos de três bytes (**FF**, **00** e **00**, nesta ordem).

(Para entender o porque dos valores armazenados, observe a representação dos valores originais, em complemento a dois!)

Atenção! Cuidado com a diferença da compactação de campos do tipo *inteiro sem sinal* (unsigned int) e *inteiro com sinal* (signed int).

Vejamos um exemplo de codificação completa. Suponha a seguinte estrutura:

```
  struct s {
    int  i;
    char s1[5];
    unsigned int u;
  };
```

e um array com duas estruturas desse tipo. Se os campos da primeira estrutura contiverem, nesta ordem, os valores -1, "abc" e 258 e os da segunda os valores 1, "ABCD" e 65535, o conteúdo armazenado para esse array será (com os valores dos bytes exibidos em hexadecimal):

```
02 21 ff 43 61 62 63 82 01 02 21 01 44 41 42 43 44 82 ff ff
```

Função `mostracomp`

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

A função `mostracomp` permite a visualização de um arquivo criado por `gravacomp` na saída padrão (terminal). Essa saída pode ser gerada, por exemplo, através de chamadas a `printf`.

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

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

- uma linha indicando o número de structs armazenadas no arquivo
- o tipo e o valor armazenado em cada campo
- uma quebra de linha deve ser inserida entre cada estrutura

Para campos do tipo *string* deve ser exibido **(str)** e a string propriamente dita (com formato %s).

Para campos do tipo *unsigned int* deve ser exibido **(uns)** e o valor original em decimal (formato %u) e, entre parênteses, em hexadecimal (formato %08x).

Para campos do tipo *signed int* deve ser exibido **(int)** e o valor original em decimal (formato %d) e, entre parênteses, em hexadecimal (formato %08x).

Para o arquivo do exemplo discutido acima, a saída de `mostracomp` seria

```
Estruturas: 2

(int) -1 (ffffffff)
(str) abc
(uns) 258 (00000102)

(int) 1 (00000001)
(str) ABCD
(uns) 65535 (0000ffff)
```

Implementação e Execução

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

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

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 `gravacomp`) ou lido (por `mostracomp`). O descritor do arquivo aberto será passado, como parâmetro, para essas funções.

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

```
gcc -Wall -o teste gravacomp.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 e vá introduzindo mais tipos de campos à medida que os casos anteriores estejam funcionando. Não esqueça de experimentar diferentes tipos de alinhamento!

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 **gravacomp.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).