INF1018 - Software Básico (2015.2) Primeiro Trabalho

Vetor de Pequenos Inteiros

O objetivo do trabalho é construir uma biblioteca para manipulação de vetores que armazenam 4 ***pequenos inteiros***, com sinal, de 6 bits. Essa biblioteca deverá oferecer operações para criação, impressão, soma, deslocamentos e persistência desses vetores.

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

Formato do Vetor de Pequenos Inteiros

Um vetor de pequenos inteiros é armazenado em 32 bits, com o seguinte formato:

bits:	**31**	**30**	**29**	**28**	**27-24**	**23-18**	**17-12**	**11-6**	**5-0**
conteúdo:	Ov 0	Ov 1	Ov 2	Ov 3	0 0 0 0	Int 0	Int 1	Int 2	Int 3

- Os bits **31** (Ov 0), **30** (Ov 1), **29** (Ov 2) e **28** (Ov 3) indicam a ocorrência de *overflow* na geração dos valores armazenados, respectivamente, nos campos Int 0, Int 1, Int 2 e Int 3 (conforme a descrição das operações, adiante).
- Os bits **27 a 24** contém o valor 0 (zero).
- Os bits **23-18** (Int 0), **17-12** (Int 1), **11-6** (Int 2) e **5-0** (Int 3) armazenam os pequenos inteiros. Cada um desses pequenos inteiros é um valor inteiro com sinal, representado em complemento a 2, com 6 bits.

Esse vetor de pequenos inteiros é representado, em C, pelo tipo **`VetSmallInt`**:

```
typedef unsigned VetSmallInt;
```

Operações da Biblioteca

- **Função `vs_new`:**

  ```
  VetSmallInt vs_new(int val[]);
  ```

  A função **`vs_new`** cria um novo vetor de pequenos inteiros. Ela recebe como argumento um vetor de inteiros com sinal (valores de 32 bits), e armazena cada um desses valores nos campos **Int 0** a **Int 3** (o primeiro elemento do vetor no campo **Int 0**, o segundo em **Int 1**, e assim sucessivamente. Essa função retorna o vetor de pequenos inteiros (`VetSmallInt`) criado.

Note que a redução da representação de um inteiro de 32 bits para 6 bits (com **truncamento**), pode alterar o valor original. Neste caso, o bit **Ov** correspondente ao pequeno inteiro deverá receber o valor 1. Caso o valor armazenado no pequeno inteiro corresponda ao valor original, o bit **Ov** correspondente deverá receber o valor 0.

Por exemplo, se o vetor de inteiros fornecido para a criação do vetor de pequenos inteiros for

```
int v1[4] = {1,-2, 3, -4};
```

o vetor de pequenos inteiros gerados será

bits:	31	30	29	28	27-24	23-18	17-12	11-6	5-0
conteúdo:	0	0	0	0	0000	000001	111110	000011	111100

Veja que os bits **Ov** correspondentes aos quatro pequenos inteiros armazenados indicam que não houve alteração nos valores fornecidos.

Se o vetor de inteiros que originará o vetor de pequenos inteiros for

```
int v2[4] = {100, 128, 55, -32};
```

O vetor de pequenos inteiros gerados será

bits:	31	30	29	28	27-24	23-18	17-12	11-6	5-0
conteúdo:	1	1	1	0	0000	100100	000000	110111	100000

Neste caso, os bits **Ov** indicam houve alteração de valor para os três primeiros inteiros.

- **Função `vs_print`:**

```
void vs_print(VetSmallInt v);
```

A função **`vs_print`** exibe o vetor de pequenos inteiros fornecido como parâmetro. Ela deve mostrar, para cada pequeno inteiro armazenado, se houve um *overflow* na geração do seu valor, e o valor armazenado em notação decimal e hexadecimal.

Veja a seguir um exemplo da saída a ser gerada para os dois vetores de pequenos inteiros mostrados acima:

```
Overflow: nao   nao   nao   nao
Valores: 1 (01) -2 (3e) 3 (03) -4 (3c)

Overflow: sim   sim   sim   nao
Valores: -28 (24) 0 (00) -9 (37) -32 (20)
```

- **Função `vs_add`:**

```
VetSmallInt vs_add(VetSmallInt v1, VetSmallInt v2);
```

A função **`vs_add`** tem como argumentos dois vetores de pequenos inteiros e retorna

um vetor de pequenos inteiros que corresponde às somas, par a par, dos pequenos inteiros armazenados nos dois vetores fornecidos (isto é, o pequeno inteiro **Int 0** do vetor gerado corresponde à soma dos pequenos inteiros **Int 0** dos dois vetores de entrada, o pequeno inteiro **Int 1** à soma dos pequenos inteiros **Int 1**, e assim sucessivamente).

Por exemplo, o vetor de pequenos inteiros correspondente à soma dos dois vetores usados nos exemplos anteriores, quando exibido, mostrará:

```
Overflow: nao   nao   nao   sim
Valores: -27 (25) -2 (3e) -6 (3a) 28 (1c)
```

Repare que a soma dos dois pequenos inteiros **Int 3** resultou em um *overflow*, e o valor dessa soma foi truncado em 6 bits para armazenamento no vetor soma.

- **Função `vs_shl`:**

```
VetSmallInt vs_shl(VetSmallInt v, int n);
```

A função **`vs_shl`** tem como argumentos um vetor de pequenos inteiros (**v**) e um valor inteiro **n**. Ela retorna um vetor de pequenos inteiros que corresponde ao deslocamento para a esquerda de **n** bits de cada um dos pequenos inteiros do vetor fornecido como parâmetro. O vetor de pequenos inteiros resultante deverá ter todos os bits **Ov** zerados.

Por exemplo, caso o vetor de pequenos inteiros fornecido for

```
Valores: -22 (2a) 21 (15) -4 (3c) 15 (0f)
```

o deslocamento de 2 bits para a esquerda resultará no vetor

```
Valores: -24 (28) 20 (14) -16 (30) -4 (3c)
```

- **Função `vs_shr`:**

```
VetSmallInt vs_shr(VetSmallInt v, int n);
```

A função **`vs_shr`** tem como argumentos um vetor de pequenos inteiros (**v**) e um valor inteiro **n**. Ela retorna um vetor de pequenos inteiros que corresponde ao deslocamento para a direita (**shift lógico**, com preenchimento de zeros) de **n** bits de cada um dos pequenos inteiros do vetor fornecido como parâmetro. Novamente, o vetor de pequenos inteiros resultante deverá ter todos os bits **Ov** zerados.

- **Função `vs_sar`:**

```
VetSmallInt vs_sar(VetSmallInt v, int n);
```

A função **`vs_sar`** tem como argumentos um vetor de pequenos inteiros (**v**) e um valor inteiro **n**. Ela retorna um vetor de pequenos inteiros que corresponde ao deslocamento para a direita (**shift aritmético**, considerando sinal) de **n** bits de cada um dos pequenos inteiros do vetor fornecido como parâmetro. Novamente, o vetor de pequenos inteiros resultante deverá ter todos os bits **Ov** zerados.

- **Função `vs_write`:**

  ```
  int vs_write(VetSmallInt v, FILE *f);
  ```

  A função **`vs_write`** tem como argumentos um vetor de pequenos inteiros (**v**) e o descritor de um arquivo aberto para a escrita, em modo binário (**f**). Essa função deverá gravar, no arquivo fornecido, o inteiro sem sinal (*unsigned int*) que representa o vetor de pequenos inteiros **em ordenação big-endian** (ou seja, o byte mais significativo deve ser o primeiro a ser gravado).

  A função deverá retornar o valor 0 se a gravação foi realizada com sucesso e -1 em caso contrário.

- **Função `vs_read`:**

  ```
  VetSmallInt vs_read(FILE *f);
  ```

  A função **`vs_read`** tem argumento o descritor de um arquivo aberto para a leitura, em modo binário (**f**). Essa função deverá ler, desse arquivo, um vetor de pequenos inteiros, armazenado conforme descrito no item anterior (isto é, em ordenação **big-endian**).

  A função deverá retornar o vetor de pequenos inteiros lido do arquivo, ou o valor 0xFFFFFFFF se ocorrer um erro de leitura.

Implementação e Execução

Você deve criar um arquivo fonte chamado **`smallint.c`** contendo as funções descritas acima e funções auxiliares, se for o caso. Esse arquivo **não** deve conter uma função `main`! Ele deve incluir o arquivo de cabeçalho `smallint.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 `vs_write`) ou lido (por `vs_read`). 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 smallint.c teste.c
```

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

```
#include "smallint.h"
```

Dicas

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

Por exemplo, você pode implementar primeiro (e testar!) as funções que criam e exibem vetores de pequenos inteiros (`vs_new` e `vs_print`). Com essas funções funcionando, vá adicionando (e testando!) as outras funções (soma, deslocamentos).

Quando todas essas operações estiverem **bem** testadas, implemente as funções de persistência. Implemente primeiro a função de gravação (`vs_write`). 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, mostrando 16 bytes por linha (a segunda coluna de cada linha, entre '|', exibe os caracteres ASCII correspondentes a esses bytes, o que, no nosso caso, não tem relevância).

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 **smallint.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 foi implementado, o que está funcionando e, eventualmente, o que não está funcionando.

Não é necessário explicar a sua implementação neste relatório. Seu código 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).

Para grupos de alunos de turmas diferentes, são necessárias duas entregas (uma de cada aluno).

Prazo

- O trabalho deve ser entregue **até a meia-noite do dia 25/09**
- 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.