# AEDS II

Exercício Prático II
Big Numbers

Entrega: 10/Mai/2016 via Moodle

## 1 Enunciado e Contexto do Problema

Boa notícia! Você foi contratado para um estágio no CERN para trabalhar numa subdivisão da pesquisa sobre aceleração de partículas. É um trabalho extremamente minucioso em que você lidará com números de precisão muito grande. A má notícia é que seu computador é o mais antigo do departamento e não é possível instalar o programa padrão para trabalhar com os dados. Seu chefe então lhe pediu para que implementasse uma biblioteca em C para lidar com tamanha precisão numérica: a biblioteca `BigNum`. A linguagem C padrão só aceita inteiros até o tipo *long long*, mas os números que você encontrará diariamente podem ser da ordem de, por exemplo, 100!, ou seja: 93.326.215.443.944.152.681.699.238.856.266.700.490.715.968.264.381.621.468.592.963.895.217.599.993.229.915.608.941.463.976.156.518.286.253.697.920.827.223.758.251.185.210.916.864.000.000.000.000.000.000.000.000. Você deve começar implementando um tipo abstrato de dados (TAD) `BigNum` que possui cinco operações principais:

- `criar`: aloca memória para uma nova variável do tipo `BigNum`
- `destruir`: libera memória alocada para uma variável do tipo `BigNum`
- `somar`: soma duas variáveis do tipo `BigNum`
- `subtrair`: subtrai duas variáveis do tipo `BigNum`
- `imprimir`: imprime o valor de uma variável do tipo `BigNum`

Para representar um `BigNum` utiliza-se uma lista de inteiros. Na notação decimal, nós temos que o número 10 é o numero 1 na casa da dezena e 0 na casa de unidade. De forma análoga, podemos implementar um número inteiro de precisao arbitrária utilizando uma lista de inteiros onde cada "casa" é representada por um inteiro. **É obrigatório implementar este TAD usando lista encadeada com alocação dinâmica**, já que não se sabe qual o valor do maior número que será computado. Use a função `malloc()` para alocar memória. O código deve estar bem modularizado e encapsulado usando as extensões `.h` e `.c`.

## 2 Entrada e Saída

A entrada deverá ser lida de um arquivo texto com uma ou mais linhas com o nome **ep2.in**. Cada linha contém dois números do tipo `BigNum` *a* e *b* separados por um espaço. Em linhas **ímpares** (assuma que a primeira linha lida é ímpar), os números devem ser **subtraídos**; em linhas **pares**,

devem ser **somados**. Para cada linha lida da entrada, o cálculo será realizado e seu programa deve escrever o respectivo resultado em um arquivo de saída chamado **ep2.out**. A última linha do arquivo de entrada contém dois inteiros iguais a zero—seu programa deve finalizar quando ler essa linha.

**Dica:** Naturalmente, se seu programa tentar ler um `BigNum` diretamente do arquivo de entrada como `int`, `double`, `long`, etc., ele não compilará. Uma maneira de se ler é interpretar cada número como uma cadeia de caracteres e alocar dígito por dígito.

### 2.1 Exemplo de Entrada (ep2.in)

```
63983516764852402937 36301672436597872316
9201526734298 7685121869
3215128729745 475989624
712476924792051427232742179675108682905639616 864728692565
0 0
```

### 2.2 Exemplo de Saída (ep2.out)

```
27681844328254530621
9209211856167
3214652740121
712476924792051427232742179675109547634332181
```

## 3 Restrições

- A implementação deve ser feita obrigatoriamente em C sem a utilização de quaisquer bibliotecas externas.
- A implementação deverá compilar sem *warnings* em ambiente Linux (Ubuntu 14.04.LTS), utilizando a seguinte linha de comando:

  ```
  gcc -Wall *.c -o ep2
  ```

- O programa deverá rodar recebendo o arquivo de entrada ep2.in como argumento passado na linha de comando:

  ```
  ./ep2 ep2.in
  ```

- A saída deve obedecer rigorosamente ao padrão especificado pois será verificada de maneira automática e representará a maior parte da nota.

## 4 Entregáveis

Crie um diretório (nomeie o diretório com seu número de matrícula) contendo seu código fonte (todos os arquivos `.c` e `.h`) e comprima-o com o seguinte comando:

```
tar zcvf <diretorio>.tar.gz <diretorio>
```

onde `<diretorio>` indica o nome do diretório criado. Submeta o arquivo `.tar.gz` via Moodle.

# 5 Critérios de Correção

A pontuação do trabalho se divide nos seguintes critérios:

- (10%) Adequação às normas de envio.
- (20%) Funcionamento correto do programa no *valgrind*, ausência de erros de execução (e.g.: *segmentation fault*), validação da entrada (e.g.: checagem se o número de argumentos passados está correto).
- (70%) Saída correta do programa.