Centro de Matemática, Computação e Cognição – UFABC
MCTA028-15 – Programação Estruturada – 3º Quadrimestre de 2023
http://professor.ufabc.edu.br/~m.sambinelli/courses/2023Q3-PE/index.html
Prof. Maycon Sambinelli

O problema

Existem muitas versões para a origem do nome Google, a famosa empresa norte-americana que se destaca, entre outras coisas, pelos seus serviços de busca na Web. Uma das explicações mais aceitas vem de fontes ligadas à Universidade de Stanford, local onde tal empresa foi criada. Essa versão da história conta que o nome pretendido era na verdade "Googol" e que um dos seus fundadores cometeu um erro de grafia e acabou escrevendo "Google". Googol é o nome dado ao número 10^{100}. A ideia dos fundadores era escolher um nome que refletisse a imensa quantidade de informação que precisaria ser processada para indexar as páginas da Internet.

Por outro lado, podiam muito bem ter sido os zimbabuanos a inventar o nome Google. Existe uma lenda na Internet que diz que a inflação anual estimada em 2008 teria sido por volta de $6,5 \times 10^{108}$% ou 650 milhões de googols (googois?). Essa lenda surgiu do equívoco de alguém ter usado a inflação estimada de janeiro a novembro (79.600.000.000%) como sendo a inflação apenas do mês de novembro e partir daí extrapolando a estimativa anual[1]. De um jeito ou de outro, seria muito legal ter uma nota de Cem Trilhões de Dólares (ainda que sejam dólares zimbabuanos).

Figura 1: Nota de Z$100.000.000.000.000.

Para se ter uma ideia do quão absurdamente grande é um googol, veja a sua comparação com outros números enormes[2]:

- Desde que ocorreu o Big Bang, "só" se passaram 17×10^{39} de ioctossegundos (1 ioctossegundo = 10^{-24} segundo).
- Juntas, todas as pessoas do mundo viveram 5×10^{11} anos, ou 17×10^{18} segundos, ou "apenas" 17×10^{42} ioctossegundos.
- A massa do universo observável é estimada entre 10^{50} e 10^{60} Kg.
- Um Googol é aproximadamente igual a 70! (70 fatorial).

[1] https://en.wikipedia.org/wiki/Hyperinflation_in_Zimbabwe
[2] Exemplos "emprestados" da Wikipedia: https://pt.wikipedia.org/wiki/Googol.

Tendo em vista este tipo de problema, é comum em sistemas computacionais precisarmos de estruturas de dados que comportem números gigantescos. A maneira mais comum é guardar estes números em variáveis do tipo *float* ou *double* que, no final das contas (sem trocadilhos), armazenam uma aproximação do número em notação científica. Isso funciona em grande parte dos casos. Contudo, em alguns ramos da computação, como a criptografia, é preciso calcular e manipular números extremamente grandes (da ordem de 10^{300} ou mais) precisamente, sem aproximações. Comumente, variáveis do tipo inteiro em C vão apenas até 64 bits (ou $2^{64} \sim= 1,8 \times 10^{19}$) e alguns compiladores mais recentes têm suporte a inteiros de até 128 bits ($2^{128} \sim= 3,4 \times 10^{38}$) mesmo em máquinas 64 bits.

Neste projeto queremos ir além. Nós vamos desenvolver uma estrutura de dados cuja única limitação para o tamanho do número a ser armazenado seja a quantidade de memória do computador. Ao final do quadrimestre nosso projeto será capaz de lidar com números inteiros, positivos e negativos, sobre os quais efetuaremos as 3 operações aritméticas básicas ($+$, $-$, $\times$).

1 Entradas e saídas

A entrada para o seu programa é fornecida via a entrada padrão (teclado do usuário) e o seu programa deve fornecer a saída na saída padrão (escrever na tela).

A entrada consiste de n casos de teste, onde $n \geq 1$. Cada caso de teste consiste de 3 linhas:

- A primeira linha de cada caso de teste consiste de um inteiro a e a segunda, de um inteiro b. Os números a e b podem ser prefixados por um sinal $+$ ou $-$ e não há espaços entre os sinais e os dígitos dos números. Também não há um limite superior nem inferior para os números a e b: há única limitação para esses números será a quantidade de memória da máquina que está rodando o nosso problema.
- A terceira linha do caso de teste consiste de um único caractere que pode ser `+`, `-` ou `*`.

Cada caso de teste (3 linhas) está fornecendo uma expressão numérica em formato pós-fixo. Exemplos: “5 3 +” , “7 3 -” . No nosso caso, cada um dos elementos da expressão estará em uma linha separada. Para cada caso de teste (conjunto de 3 linhas), você deve fornecer uma única linha contendo o resultado da operação aritmética, efetuando a operação da terceira linha sobre os dois números que a antecederam. Caso o resultado seja um número negativo, o sinal `-` deve ser prefixado aos dígitos do número (sem a inserção de espaço entre o sinal e o número).

Observação você deve seguir o formato de saída rigorosamente e não deve imprimir nenhum texto na tela que não tenha sido especificado acima, pois a correção dos trabalhos será feita de forma automatizada.

A seguir, apresentamos alguns exemplos de entradas e saídas.

1.1 Exemplo 1

Entrada

```
3
5
+
2
8
-
12157865549787498543194363453256756324532235218789765476
3
-
```

```
3245899634153843978545123478312549796317362134
98312663286465303725297340976915494785435245565384217 8
+
9831266361105526714068173883142784261669022519712043 12
-1
-
```

Saída

```
8
-6
12157865549787498543194363453256756324532235218789765473
9831266361105526714068173883142784261669022519712043 12
9831266361105526714068173883142784261669022519712043 13
```

1.2 Exemplo 2

Entrada

```
-123582
2
*
979879888347697865634597873459723495879874529348798273458797987324524579872523 4
398798572345987528457982734598798723495879872398458823459879872359887234 58
+
3443
987987412341212341
-
```

Saída

```
-247164
979883876333421325509882453287069483867109488147522258047032586123248178744869 2
-987987412341208898
```

1.3 Outros casos de teste

Outros casos de teste podem ser obtidos no link abaixo:

http://professor.ufabc.edu.br/~m.sambinelli/courses/2023Q3-PE/trabalho/instances.tar.gz

Os arquivos `.in` contém os casos de testes de entrada e os arquivos `.out` as saídas. Para usar de forma efetiva e simples esse gabarito fornecido, ao executar o seu programa, faça redirecionamento da entra e saída padrão [3] e use um programa de comparação de arquivos como: `diff`, `delta` [4] ou `meld` [5].

[3] https://www.ppgia.pucpr.br/pt/arquivos/techdocs/linux/foca-iniciante/ch-redir.html
[4] https://github.com/dandavison/delta
[5] https://meldmerge.org/

2 Entrega

Este projeto poderá ser feito em grupos de até três pessoas. A entrega deverá ser feita pelo Moodle **apenas** por um dos integrantes do grupo. O prazo final para entrega é às 23h59m do dia **10-12-2023**.

1. Um único arquivo chamado `xxxxx.zip` deve ser entregue, onde `xxxxx` (aqui e no restante da seção) deve ser substituído pelos ra's dos participantes do grupo, separados por *underline*.
2. Ao descompactar o arquivo `xxxxx.zip`, uma única pasta chamada `xxxxx`, contendo todos os arquivos fontes do seu projeto, deve ser criada.
3. Dentro da pasta `xxxxx`, obrigatoriamente devem existir os seguintes arquivos:
   - `bignumber.h` um arquivo de *header* de C contendo toda a interface pública do seu tipo BigNumber.
   - `bignumber.c` um arquivo de C contendo a implementação da sua interface pública.
   - `client.c` um arquivo de C contendo a função `main()` e que é responsável por usar a sua biblioteca `bignumber.h` para resolver o problema do projeto.
   - `makefile` um arquivo de configuração que permita o programa `make` compilar corretamente o seu programa quando o seguinte comando for digitado dentro do diretório `xxxxx`: `make`.
   - Um arquivo `pdf` chamado `README` com no máximo 2 páginas contendo um pequeno relatório. Este relatório deve conter o nome de ra de todos os membros do grupo. Além disso, deve conter:
     - Uma explicação de como foi representado o `BigNumber`.
     - Qual a interface pública do seu tipo `BigNumber` (basta as assinaturas).
     - Mencione qualquer algoritmo ou estrutura de dados avançada que tenha sido empregada para melhorar o tempo de execução do seu código. Além disso, diga como/onde usou.
     - Diga, de forma geral, como foi a divisão de trabalho dentro da equipe, i.e., quem fez o quê?
   - Não há nenhum problema em dividir o seu programa em outros arquivos, essa é apenas a divisão mínima.
4. O seu makefile deve compilar o seu programa usando o programa `gcc` com as seguintes flags: `-std=c99 -Wall -Wextra -Wvla -g`

3 Avaliação

Para a composição da nota final do trabalho, serão levados em contas os seguintes aspectos:

- Correção das operações de soma, subtração e multiplicação.
- Desempenho do programa (tempo de execução).
- Vazamento de memória: vazamentos de memória serão verificados com o seguinte comando `valgrind --leak-check=yes cliente`. Programas com vazamento de memória serão penalizados.

- Organização e legibilidade do código: indentação correta, modularização (i.e., divisão em funções, em arquivos, criação de tipos e etc.), nomenclatura adequada (para variáveis e funções) e comentários.

Lembre-se: Plágios serão severamente punidos com a reprovação na disciplina.

3.1 Política de atrasos

Não serão aceitos trabalhos fora do prazo.

3.2 Bônus

3.2.1 Rinha de programas

Os membros do grupo do programa correto (que passar em todos os casos de teste e sem vazamentos de memória) mais rápido da turma receberão uma **nota bônus de 1 ponto na média final!**.

Para pensar em como melhorar o seu programa (e surrar o programa dos seus colegas), veja a Seção 4.

3.2.2 Git

Grupos que usarem de maneira adequada o `git` ao longo do desenvolvimento do trabalho receberão um **bônus de .5 ponto na P2**.

O `git` não será ensinado ao longo do curso, mas é uma ferramenta essencial para qualquer trabalho de desenvolvimento. Existem diversos materiais na web (livros, post em blogs, vídeos no youtube) ensinando o seu uso.

Caso o grupo opte por usar o `git` em conjunto com um serviço de hospedagem como GitHub, GitLab ou BitBucket, lembre-se de criar um repositório privado, para que os outros colegas do curso não encontrem o seu trabalho na plataforma.

Para receber esse ponto bônus, a pasta contida no arquivo zip deve ser um repositório git.

4 Dicas

Existem maneiras muito elaboradas e eficientes para representar números grandes em computadores. Uma das mais empregadas (senão a mais) chama-se *complemento de dois*. Apesar da sua eficiência, sugerimos que neste trabalho você fique longe desta técnica e aborde o problema de uma maneira muito mais simples.

Nossa sugestão é que você armazene os dígitos do numerão em um vetor, por exemplo, de inteiros. Assim, o `BigNumber` 12345 poderia ser representado pelo vetor $[5, 4, 3, 2, 1]$. Note que sugerimos que no vetor a ordem dos dígitos seja dada dos dígitos menos significativos para os mais significativos. Isso facilita na hora de fazer a soma, por exemplo, pois o algoritmo da soma tradicional (papel e lápis) trabalha dos dígitos menos significativos para os mais significativos. Por outro lado, isso força você a escrever um código que varre o vetor na ordem inversa à tradicional para imprimir o numerão (pois a ordem dos dígitos será a inversa da tradicional). Vá pensando nesse caso!

Sugerimos que você crie um tipo de dados `BigNumber` para armazenar os dígitos e qualquer outro metadados necessário pelo numerão. Isso facilitará na modularização do seu código e na definição de boas, úteis e versáteis interfaces. Uma boa modularização também levará a uma menor ocorrência de bugs e uma melhor divisão de trabalho na equipe.

Sugerimos também que você crie funções para criar um `BigNumber` a partir de um inteiro (`int`) ou a partir de uma string (`char []`) e para imprimir o `BigNumber` armazenado. Isso facilitará bastante os seus testes.

Crie funções para cada operação (soma, subtração e multiplicação) que será realizada sobre os `BigNumbers`. Recomendamos (porém a escolha é sua) que você faça funções que alterem um dos dois `BigNumber` envolvidos na operação, ao invés de criar um novo `BigNumber` apenas para armazenar o resultado da operação (isso fará com que o seu programa rode mais rápido). Neste caso você provavelmente irá necessitar de uma função de cópia entre `BigNumbers`.

Para a implementação das operações propriamente ditas, utilize os algoritmos de soma e subtração que você está acostumado (papel e lápis). Eles são surpreendentemente eficientes, mesmo para números extremamente grandes.

Repare que existem algumas dependências entre as operações. Por exemplo, para implementar a multiplicação você vai precisar da soma. Para multiplicação, existem algoritmos mais eficientes (mais rápidos) do que aquele feito "à lápis e papel". Um algoritmo melhor e razoavelmente fácil de implementar é o Karatsuba [6], existem outros com diferentes graus de complexidade.

Evite duplicar dados e tome cuidado com a passagem de parâmetros. Em C, toda passagem de parâmetro é por valor, ou seja, uma cópia do dado é feita. Uma forma de mitigar esse problema é passar endereços de memória das estruturas grandes (um endereço de memória é um dado pequeno – 64 bits). Copiar um `struct` com um vetor estático como membro requer uma operação de cópia de todos os elementos do vetor (se o vetor foi alocado dinamicamente, então apenas endereços são copiados).

Você pode usar o programa `gprof` [7] para analisar o tempo gasto em cada etapa de execução do seu programa, para tentar identificar gargalos e, assim, estudar melhorias para otimizar esses pontos. O uso do `gprof` não será ensinado ao longo do curso. É uma ferramenta simples e existem diversos bons materiais na web ensinando a usá-lo.

Você pode verificar vazamentos de memória no seu programa com o uso do programa `valgrind`. O professor Rafael Schouery escreveu um tutorial bem simples sobre o seu uso[8].

4.1 Sugestão de passos para o desenvolvimento

- Comece trabalhando apenas com números positivos.
- Implemente a operação de soma e teste-a à exaustão!
- Faça a subtração. Primeiramente trabalhe apenas os casos de $a - b$ onde $a > b$. Em seguida você vai ter que adicionar o suporte aos números negativos. Teste à exaustão!
- Faça testes exaustivos na soma com números negativos e positivos. Perceba que:
  - $a + b$ com $a \geq 0$ e $b < 0$ é igual a $a - |b|$.
  - $a - b$ com $b < 0$ é igual a $a + |b|$.
  - $a + b$ com $a < 0$ e $b < 0$ é igual a $-(|a| + |b|)$.
  - ...
- Note que utilizando essas transformações é possível utilizar o código que você já tinha feito apenas para números positivos para tratar o caso de números negativos também!

Divirta-se!

[6] https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/karatsuba.html
[7] https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_mono/gprof.html
[8] https://www.ic.unicamp.br/en/~rafael/materiais/valgrind.html