MC202 — Estruturas de Dados

Laboratório 3 - Googol

Primeiro semestre de 2017 - Turmas B e C
Professor: Emilio Francesquini
francesquini@ic.unicamp.br

1 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 por um erro de grafia cometido por um dos seus fundadores ele acabou escrevendo "Google". Um 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.

[1] https://en.wikipedia.org/wiki/Hyperinflation_in_Zimbabwe

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).

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 (ordem de 10^{300} ou mais) precisamente, sem aproximações. Comumente variáveis do tipo inteiro em C vão apenas até 64 bits (ou $2^{64} \sim= 1,8\times10^{19}$) e, alguns compiladores mais recentes, tem suporte a inteiros de até 128 bits ($2^{128} \sim= 3,4\times10^{38}$) mesmo em máquinas 64 bits.

Neste laboratório 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. A nossa estrutura de dados deverá ser capaz de representar números inteiros, positivos e negativos, capaz de efetuar as 4 operações aritméticas básicas (+, -, $\times$, $\div$) além de calcular o resto da divisão.

2 Organização do código

Você pode estruturar o seu programa da maneira que achar mais conveniente para cumprir a tarefa dada. Contudo, aqui estão algumas sugestões que podem ajudar.

2.1 Estrutura de dados

Crie uma estrutura de dados para representar o seu número. Crie também funções para criar um número a partir de um número inteiro (`int`) ou a

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

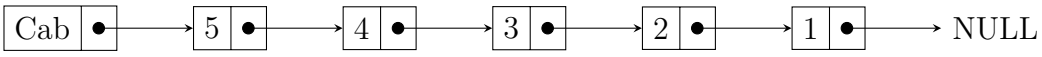
partir de uma string (`char*`). Note que você também vai precisar das funções para se desfazer dessas estruturas já que você vai precisar alocar e desalocar memória para elas.

```
struct _numerao {
    ...//conteúdo da sua estrutura
};
typedef struct _numerao numerao;

numerao *criaNumeraoDeInt(int n) {…};
numerao *criaNumeraoDeString(char *str) {…};
void destroiNumerao(numerao *n) {…};
void imprimeNumerao(numerao *n) {…};
```

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

Minha sugestão é que você represente os dígitos do número que você quer armazenar como inteiros de uma lista ligada. Por exemplo, o numero 12345 poderia ser representado pela seguinte lista ligada:



Note que na lista a ordem dos dígitos é dada dos dígitos menos significativos para os mais significativos. Isso facilita na hora de fazer a soma, por exemplo, pois os algoritmo da soma tradicional (papel e lápis) trabalha dos dígitos menos significativos para os mais significativos. Por outro lado, isso atrapalha um pouco na hora de escrever o código para imprimir os números (pois a ordem dos dígitos será a inversa da tradicional) ou para fazer a divisão (já que o algoritmo trabalha com os dígitos mais significativos do dividendo primeiro). Neste caso uma lista duplamente ligada pode ajudar.

Em resumo, um rascunho para a estrutura de dados sugerida seria:

```
struct _numerao {
    sinal; //O número representado é positivo ou negativo?
    cabeça da lista ligada de dígitos;
    cauda (se a lista for duplamente ligada) da lista de dígitos;
};
```

2.2 Operações

Será necessário criar uma função para cada operação em cima da sua estrutura de dados. Eu recomendo (porém a escolha é sua) que você faça funções que alterem o valor passado como parâmetro e não funções que devolvem um novo número com o resultado da operação. Isso facilita o controle de uso de memória da sua aplicação. Neste caso você provavelmente também vai querer criar uma função de cópia que devolve uma cópia do número passado como parâmetro.

```
//Soma a e b e guarda o resultado (a+b) na
//estrutura de dados apontada por 'a'
//'b' não é modificado
void soma(numerao *a, numerao *b) {…};

//Subtrai a de b e guarda o resultado (a-b) na
//estrutura de dados apontada por 'a'
//'b' não é modificado
void subtrai(numerao *a, numerao *b) {…};

//Multiplica a por b e guarda o resultado (a×b) na
//estrutura de dados apontada por 'a'
//'b' não é modificado
void multiplica(numerao *a, numerao *b) {…};

//Divide a por b e guarda o resultado (a÷b) na
//estrutura de dados apontada por 'a'
//'b' não é modificado
void divide(numerao *a, numerao *b) {…};

//Divide a por b e guarda o resto da divisão (a%b) na
//estrutura de dados apontada por 'a'
//'b' não é modificado
void restoDivisao(numerao *a, numerao *b) {…};

//Aloca e devolve um ponteiro para um numerao cujo
//conteúdo é idêntico ao numerao recebido como parâmetro
//'a' não é modificado
numerao *copiaNumerao(numerao *a) {…};
```

Exemplo de uso:

```
//Cria dois números baseados em ints
numerao *a = criaNumeraoDeInt(4);
numerao *b = criaNumeraoDeInt(3);
soma (a, b); //O resultado é colocado no próprio 'a'
imprimeNumerao(a); //Imprime 7
numerao *c = copiaNumerao (a);
divide (c, 3);
imprimeNumerao(a); //Imprime 7
imprimeNumerao(c); //Imprime 2
//Libera a memória utilizada por 'a', 'b' e 'c'
destroiNumerao(a);
destroiNumerao(b);
destroiNumerao(c);
```

Lembre-se, sua estrutura de dados trabalha somente com inteiros. Divisões devem ser truncadas. Ex: $7 \div 2 = 3$, $-5 \div 3 = -1$.

Para a implementação das operações propriamente ditas, utilize os algoritmos de soma, subtração e divisão que você está acostumado. Aqueles mesmos que você usa quando você faz a conta no papel. Eles são surpreendentemente eficientes, mesmo para números extremamente grandes.

Logo no início da sua implementação você vai reparar que existem algumas dependências entre as operações. Por exemplo, para implementar a multiplicação você vai precisar da soma. Para implementar a divisão você vai precisar de subtração. Para implementar a criação de um número a partir de um inteiro você vai precisar da soma e da multiplicação prontas, que por sua vez dependem de você já ter criado um número a partir de algum lugar. Essa dependência circular pode ser quebrada na mão, criando os primeiros números para os seus testes manualmente, sem o auxílio de funções de criação. Comece simples, com números de apenas um dígito e vá construindo o seu programa a partir daí.

Minha **sugestão** para o desenvolvimento é:

- Comece trabalhando apenas com números positivos
- Implemente a operação de soma usando números construídos (a lista ligada) na mão. Teste-a à exaustão!
- Implemente a multiplicação. Primeiramente com números de um só

dígito. Depois generalize para números de múltiplos dígitos. Teste novamente 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. Faça testes exaustivos na soma e multiplicação com números negativos e positivos. Perceba que:
  - soma (a, b) com a positivo e b negativo = sub(a, valorAbsoluto(b))
  - subtrai(a, b) com b negativo = soma (a, valorAbsoluto(b))
  - soma (a, b) com ambos a e b negativos = -soma (valorAbsoluto(a), valorAbsoluto(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! Coloque logo no início da sua função de subtração, por exemplo:

```
numerao *subtrai (numerao *a, numerao *b) {
    if (b é negativo) {//b = -c, a-b = a - (-c) = a + c
        muda o sinal de b para positivo;
        soma (a, b);
        para manter b inalterado volta o sinal para negativo;
        return a
    }
    …
```

  Note que na função acima, caso `a` também seja negativo, a função soma pode aplicar novamente uma das transformações que discutimos e chamar a função `subtrai` de volta para fazer a conta `subtrai(b, valorAbsoluto(a))`, ou seja, altere os parâmetros para eles "caberem" nas suas implementações que só funcionavam com números positivos.

- Só depois que tudo acima estiver funcionando perfeitamente parta para a divisão e resto. Esta é a operação mais complicada de todas e depende de tudo o que você fez antes. Então, para evitar dores de cabeça repito: teste com afinco as outras três operações!

3 Entradas e saídas

O seu programa deve ler os dados da entrada padrão (stdin) e escrever os resultados na saida padrão (stdout). Para isto você pode usar as funções `scanf` e `printf`.

- A entrada será dada por uma expressão numérica dada em formato pós-fixo.
- Cada um dos elementos da expressão estará em uma linha separada.
- Cada linha de entrada trará um número (tamanho arbitrário), uma operação (+, -, *, /, %) ou "FIM". Por conveniência vamos limitar os números a 1000 dígitos (ou seja, 1001 caracteres no máximo quando incluirmos números negativos)
- Note que operações feitas com dois números de 1000 dígitos podem resultar em números com mais de 1000 dígitos. O limite de 1000 dígitos é apenas válido para os números dados como entrada. Mas não há limite quanto ao tamanho dos números obtidos como resultado de operações.
- Você pode assumir que a entrada é sempre uma expressão válida.
- Seu programa deve parar a execução quando receber a entrada "FIM"
- Você também pode assumir que o número máximo de operandos empilhados é limitado a 10.
- Toda vez que uma operação for lida, ela deve ser aplicada aos dois elementos do topo da pilha e o resultado da operação empilhado e impresso na tela.

O modo de funcionamento do programa é bem semelhante à uma calculadora com notação pós-fixa. Veja um exemplo de calculadora que roda dentro do Google Chrome aqui `https://chrome.google.com/webstore/detail/rpn-calculator/iblaobcdcjnenokjemcodanggahddfnd`.

Exemplo 1: (entrada em verde saída em vermelho)

```
 3
5
+
8
2
/
```

```
4
3
%
1
5
*
5
-1
-
6
FIM
```

Exemplo 2:

```
2
3
4
5
6
7
8
9
10
*
90
*
720
*
5040
*
30240
*
151200
*
604800
*
1814400
*
3628800
FIM
```

Exemplo 3:

```
98745265432294533098524308943230863 22
32514228756483735474233118787209009871
*
32106261488853302669357990544032565171141269530189428930773981431848308446 29
/
35673623876503669632619989493369516856823632811321587700859979368720342718
FIM
```

4 Entrega e avaliação

Este laboratório poderá ser feito em duplas. A entrega deverá ser feita pelo Susy **apenas** por um dos integrantes da dupla. O arquivo enviado deve conter um cabeçalho contendo o nome completo e RA de cada um. Verifique a página da disciplina (`http://www.ic.unicamp.br/~francesquini/mc202/`) para informações sobre as datas de entrega e acesso ao Susy.

Todos os trabalhos entregues passarão por uma correção automática no Susy. A nota será proporcional ao número de testes que passarem com sucesso. Contudo, os monitores e o professor poderão, ao seu próprio critério, alterar as notas para cima ou para baixo. Passar na metade do testes não significa que você tirou nota 5 automaticamente, indica apenas a provável nota que será atribuída ao seu trabalho.

ATENÇÃO: Plágios serão severamente punidos com a reprovação na disciplina.

Bom trabalho!