ICC 2 – Trabalho 3

Leiam com atenção. Receberemos questões para montar o FAQ do Trabalho 03 até o dia 15 de agosto de 2013.

Entrega até o dia 19/08/2013 às 23:59 h.

Utilizando a linguagem C, considere um arquivo texto cujo nome será dado como entrada para seu programa. Leia o nome desse arquivo utilizando a função scanf. A seguir, crie um ponteiro para armazenar todo o conteúdo desse arquivo em memória Heap (http://pt.wikipedia.org/wiki/Gerenciamento_de_mem%C3%B3ria).

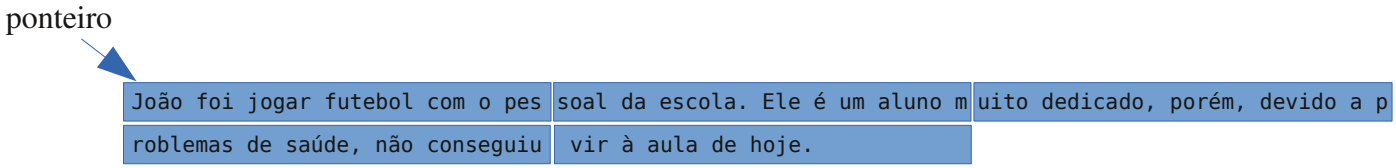
Faça então a leitura do arquivo texto (pode ler um caracter por vez, ou um grupo de caracteres por vez), e utilize a função realloc para realizar a alocação da memória necessária, até armazenar todo o conteúdo do arquivo texto em memória Heap. Assim, você terá uma região de memória dinamicamente alocada para guardar todo o conteúdo do arquivo.

Tarefa:

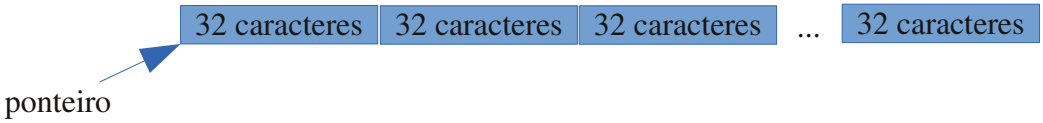
1) Utilizando o ponteiro que você criou (e que aponta para o texto lido do arquivo), considere cada grupo de 32 caracteres como um número inteiro de 32 bytes (o que totaliza 256 bits por grupo). Exemplo:

Considere que o arquivo texto contém:

```
João foi jogar futebol com o pessoal da escola. Ele é um aluno muito dedicado, porém, devido a problemas de saúde, não conseguiu vir à aula de hoje.
```



Em termos de caracteres temos:



Observe que o último grupo de 32 caracteres contém apenas 20 caracteres, contando um espaço que está localizado logo no ínicio desse grupo. Para compreender melhor:

“ vir à aula de hoje.”

Neste caso, sempre que um grupo de 32 caracteres não tiver totalmente preenchido, complete ele com o caracter de espaço (' '), obtendo algo como:

“ vir à aula de hoje. ”

Observe que, para isso, você deverá alocar mais memória Heap utilizando o comando realloc.

Agora considere que cada grupo de 32 caracteres forma um número inteiro de 32 bytes, ou seja, um número inteiro com 256 bits. Em seguida, calcule a soma de todos os números inteiros de 32 bytes.

Essa soma deve ser armazenada na forma de um numero inteiro de 256 bits. Com certeza haverá estouro (*overflow*) dos números resultantes. Você deverá desconsiderar esse estouro. Por exemplo:

> *Considere que cada grupo tem apenas 8 bits apenas para fins de compreensão do problema de estouro. Ao somar dois números de 8 bits tal como abaixo (Lembre-se que em seu programa serão 256 bits):*
>
> ```
>  11111111
> +
>  11111111
> ---------
> 111111110
> ```
>
> *Observe que o resultado tem 9 bits, i.e., há um bit a mais à esquerda. Neste exemplo definimos que iremos utilizar apenas 8 bits, assim devemos desconsiderar o bit mais significativo (ou seja, o bit mais à esquerda), produzindo o resultado* `11111110` *em binário.*

Seu programa deverá imprimir na tela 256 bits relativos à soma dos grupos de caracteres. Esse resultado deverá ser escrito como saída na forma de bits.

> *O princípio adotado neste trabalho segue um conceito similar ao adotado no algoritmo MD5sum (http://en.wikipedia.org/wiki/Md5sum) para produzir uma sequência de bits que represente um arquivo. Esse recurso é muito utilizado para sabermos quando um arquivo foi alterado. Por exemplo, calcula-se o MD5sum de um arquivo texto em uma data e 5 dias depois. Se o resultado do MD5sum alterou, o arquivo texto foi alterado e, por exemplo, podemos proceder com o backup desse arquivo (pois sabemos que não temos uma versão dele ainda salva em HD externo de backup).*
>
> *Essa mesma ideia é utilizada para verificar arquivos corrompidos, inclusive para evitar falhas de segurança em sistemas computacionais.*
>
> *Se deseja testar o MD5sum, execute em seu Linux:*
>
> *1) Crie um arquivo chamado meuarquivo.txt e inclua algum texto nele*
> *2) Execute: md5sum meuarquivo.txt*
> *3) Altere o conteúdo do arquivo*
> *4) Execute novamente: md5sum meuarquivo.txt*
> *5) Compare os resultados do md5sum*

Formato da entrada:
`<nome_do_arquivo>`

Formato da saída:
`<256_bits>\n`

A saída NÃO DEVE ser impressa para Tarefas não solicitadas!

```
Arquivo.txt
--------------------------------------------------------------------------------
João joga futebol com o pessoal da escola desde o ano passado. Ele é um aluno
muito dedicado.
```

Exemplo 1

Entrada fornecida:
```
Arquivo.txt
```

Saída esperada:
```
1011000001000000100010011110010010011000001111001110111001010100010000001110001010
1000100011011101010000010101100011100011100111111111101010100001010101000111001010 0
1100111111001010010101000100111001100100110100111100001101100011110001001101101111
0110101100
```

Informações importantes (LEIA COM ATENÇÃO)

- Sobre a avaliação

  1. Um dos objetivos da disciplina de ICC2 é o **aprendizado individual** dos conceitos de programação. A principal evidência desse aprendizado está nos trabalhos, que são individuais neste curso. Você deverá desenvolver seu trabalho sem copiar trechos de código de outros alunos, nem codificar em conjunto. Portanto, compartilhem idéias, soluções, modos de resolver o problema, mas não o código.
     - O plágio vai contra o código de ética da USP.
     - Quando autores e copiadores combinam, estão ludibriando o sistema de avaliação.
     - O trabalho em grupo e a cooperação entre colegas é em geral benéfico e útil ao aprendizado. Para ajudar um colega você pode lhe explicar estratégias e idéias. Por exemplo, pode explicar que é preciso usar dois loops para processar os dados, ou que para poupar memória basta usar uma certa estrutura de dados, etc. O que você **não** deve fazer é mostrar o seu código. Mostrar/compartilhar o código pode prejudicar o aprendizado do seu colega:
       - depois de o seu colega ter visto o seu código, será muito mais difícil para ele imaginar uma solução original e própria;
       - o seu colega não entenderá realmente o problema: a compreensão passa pela prática da codificação e não pela imitação/cópia.
     - Um colega que tenha visto a sua solução pode eventualmente divulgá-la a outros colegas, deixando você numa situação muito complicada, por tabela.
     - O texto acima foi baseado e adaptado da página http://www.ime.usp.br/~mac2166/plagio/, da qual recomendo a leitura completa.

  2. Todos os códigos fontes serão comparados por um (ou mais) sistema(s) de detecção de plágio, e **os**

trabalhos com alta similaridade detectada terão suas notas zeradas, tanto aqueles relativos ao código de origem quanto do código copiado. **A detecção de plágio será reportada à Seção de Graduação para providências administrativas**.

3. A avaliação incluirá a porcentagem de acertos verificada pelo SQTPM e também a análise do seu código, incluindo endentação, comentários, bom uso da memória e práticas de programação. Portanto faça seu código com cuidado, da melhor forma possível.

- Sobre o sistema de submissão:

1. Seu código deverá incluir arquivo fonte .c .h e Makefile – todos os arquivos deverão **obrigatoriamente** conter no início um comentário com seu nome, número USP, turma e data da entrega do trabalho.

2. **A data/hora de entrega do trabalho é aquela estipulada no sistema. Trabalhos entregues por email NÃO serão aceitos, mesmo que dentro da data/hora estipulada. Faça seu trabalho com antecedência para evitar entregar em cima da hora e ter problemas de submissão, pois o sistema tende a ficar lento com as múltiplas submissões feitas geralmente próximas ao fechamento do sistema.**
    - A submissão é de responsabilidade do aluno, e os problemas comuns à entrega próxima ao fechamento do sistema também. Portanto: problemas de acesso à rede **não** serão aceitos como desculpa para entrega por email ou fora do prazo.

3. A compilação e execução do código é feita no sistema pelos comandos:
```
make all
make run
```

4. A saída do seu programa deve ser exatamente igual à saída esperada, incluindo: espaços em branco, quebras de linha e precisão decimal.

5. Há um limite em segundos para a execução dos casos de teste e um limite de memória total para ser utilizada. Você deverá gerenciar bem o tempo de execução e o espaço ocupado para que seu programa fique dentro desses limites, para evitar uso excessivo, pois o sistema irá invalidar o caso em que o limite foi excedido.

6. Ao enviar, aguarde **sem recarregar a página nem pressionar ESC**, para obter a resposta. Caso demore mais do que 2 minutos para dar uma resposta, feche e abra novamente a página do servidor. Verifique se seu programa tem algum problema de entrada de dados (ou se tem algum loop infinito ou parada para pressionamento de tecla). Caso não tenha, aguarde alguns minutos e tente novamente.

7. O erro de "Violação de Memória" significa acesso indevido a arranjo ou arquivo. Use compilação com **-g** e o programa valgrind para tentar detectar a linha de código com erro.

    - Exemplo 1 de violação de memória:
```
int **mat = (int **)malloc(sizeof(int *) * 3);
mat[0] = (int *)malloc(sizeof(int)*10);
for (i = 1; i < 3; i++) {
   free(mat[i]);
}
// apenas a posicao 0 de mat foi alocada as outras nao
```

```
   // portanto esta liberando regiao nao alocada
   // gerando violacao de memoria
```

- Exemplo 2 de violação de memória:

```
int B[5] = {5, 6, 7, 8, 9};
int N = 5;
int *A = malloc(N*sizeof(int));
int j = 0, i = 1; // 'j' inicializado em 0, 'i' inicializado em 1
while (j < N){
      A[i] = B[j]; // 'j' e 'i' sao indices diferentes
      j++;         // quando j = (N-1) i = (N-1)+1 e
      i++;             // havera escrita indevida em A
}
```