Departamento de Ciência da Computação – IME-USP

MAC 115 – Introdução à Computação para Ciências Exatas e Tecnologia

Escola Politécnica – Primeiro Semestre de 1997

Quarto Exercício-Programa — Data de entrega: até 19 de junho de 1997.

Turmas: Produção, Civil A, Civil C, Minas/Metal/Materiais

TRANSFORMAÇÕES EM UM ARQUIVO DE TEXTO

Neste exercício-programa você vai escrever em C um programa que utiliza as funções que você escreveu no terceiro EP, especialmente a função **substitui_todas** (as outras funçoes não serão usadas diretamente pelo programa, mas sim indiretamente, pois a função **substitui_todas** as utiliza). O programa vai ler um arquivo de texto linha por linha, regravando cada linha após operar nela uma certa transformação, a saber : a substituição de todas as ocorrências de um string **t** nesta linha, por outro string **r**, onde **t** e **r** são dados pelo usuário. Então o programa opera da seguinte forma :

(1) pede ao usuário o nome do arquivo de entrada;

(2) pede ao usuário o nome do arquivo de saída;

(3) pede ao usuário o string **t** a substituir;

(4) pede ao usuário o string **r** substituidor;

(5) abre o arquivo de entrada — para isto, você usa algo como : **ent = fopen(nome_input,"r")**, onde **ent** é uma variável introduzida por **FILE *ent**, e **nome_input** é um string que contém o nome do arquivo de entrada dado no item (1);

(6) abre o arquivo de saída — para isto, você usa algo como : **sai = fopen(nome_output,"w")**, onde **sai** é uma variável introduzida por **FILE *sai**, e **nome_output** é um string que contém o nome do arquivo de saída dado no item (2);

(7) lê uma linha do arquivo de entrada — para isto, você pode usar **fgets (linha, tam, ent)**, onde **linha** é o string que receberá uma linha lida do arquivo, **tam** é um limite para o tamanho da linha lida (use 100, por exemplo), e **ent** é aquele apontador que você recebeu da **fopen** (item 5);

(9) substitui as ocorrências de **t** na linha lida por ocorrências de **r** — para isto, basta usar a função **susbstitui_todas**;

(11) grava a linha no arquivo de saída — para isto, você pode usar **fputs (linha, tam, sai)**, onde **linha** é o string onde está a linha a gravar no arquivo, **tam** é um limite para o tamanho da linha a gravar (use 100, por exemplo), e **sai** é aquele apontador que você recebeu da **fopen** (item 6);

(12) os passos (7)-(11) são repetidos enquanto o arquivo de entrada não acabar (a função **fgets** retorna 0 quando acabar o arquivo, ou seja, quando todas as suas linhas já tiverem sido lidas). Portanto, o laço principal de seu programa será algo como

while (fgets(...) != 0) { transforma_linha (); grava_linha ();}

(13) fecha os arquivos de entrada e de saída.

Mas então o EP4 é só isso ??? Não exatamente, pois ainda tem um detalhe...

Você deve ter notado que na numeração acima faltaram os itens (8) e (10) (se não notou, precisa prestar mais atenção nas numerações ...).

O detalhe é o seguinte : existe um modo de se guardar arquivos que chamaremos de *modo comprimido*. Este modo consiste em empregar alguma técnica que permita representar dados usando menos bytes do que seriam necessários para representar estes mesmos dados no modo normal, ou seja, alguma técnica através da qual possamos "abreviar" os dados. A técnica que empregaremos aqui é a mais simples possível, mas ainda assim nos permitirá economizar preciosos bytes de disco, se o arquivo tiver *sequências uniformes*, isto é, sequências de um mesmo caracter (por exemplo, sequências de brancos). Ela consiste em economizar posições que seriam ocupadas por sequências formadas de pelo menos 4 caracteres iguais consecutivos, substituindo tais sequências por apenas 3 caracteres, a saber :

(i) o caracter que se repete na sequência;

(ii) um caracter especial para indicar que o primeiro caracter deve ser repetido — escolheremos o caracter '#' para este papel;

(iii) um caracter que representa o número total de caracteres da sequência **somado com 32**. Assim, se este caracter corresponder ao número 43 na tabela ASCII, estará indicando uma sequência de tamanho 11 (= 43 - 32). O professor poderá lhe explicar em aula a razão deste "deslocamento" de 32.

Como exemplo desta técnica, observe o seguinte string :

											h	a	j	a		b	r	a	n	c	o	s	!	!	!	!	!	!	!	!	NUL	

No string acima temos 2 sequências: uma de brancos e outra de exclamações. Vamos colocá-lo no modo comprimido, usando a convenção descrita acima. Segundo esta convenção, o string acima pode ser representado por

	#	+	h	a	j	a		b	r	a	n	c	o	s	!	#	(	NUL														

Repare como o string, assim representado, passou a ocupar menos posições do vetor. Note que o caracter '+' que vem logo após o primeiro '#' indica tamanho 11 (eram 11 brancos na sequência original), pois seu valor na tabela ASCII é 43, e 43 - 32 = 11. Por sua vez, o caracter '(' que vem logo após o segundo '#' indica tamanho 8 (eram 8 exclamações na sequência original), pois seu valor na tabela ASCII é 40, e 40 - 32 = 8.

Você deve estar se perguntando: e como vou saber que caracter corresponde a que número, para poder marcar o tamanho da sequência após um '#' que indica sequência uniforme ? **Você não precisa saber !** Na verdade, como você deve ter aprendido em aula, caracteres podem ser tratados como números inteiros. O que você precisa fazer, ao converter uma linha do modo normal para o modo comprimido, é colocar após o '#', no vetor de caracteres, o **número** 32+n, onde n é o tamanho da sequência sendo representada. Analogamente, ao converter uma linha do modo comprimido para o modo normal, o caracter que vem após o '#' deve ser tratado como um número, do qual você subtrai 32 para obter o tamanho da sequência original representada.

Você também deve estar se perguntando: então o arquivo de texto não vai poder ter o caracter '#' "propriamente", isto é, valendo por ele mesmo ? Isto é fácil de resolver — é só adotar mais a seguinte convenção: se num string o caracter '#' for encontrado, seu significado depende do caracter que vem logo após ele:

(i) se for outro '#', então os dois '#' representam apenas um '#' mesmo (é como usar '%%' para representar um '%', em scanf e printf, escapando de seu significado como indicador de formato);

(ii) se for outro caracter, então trate-o como indicador do tamanho de uma sequência uniforme com o caracter que vem antes do '#'.

O seu programa, portanto, vai lidar com dois tipos de arquivos :

(1) o arquivo de texto normal, cujo nome, **obrigatoriamente**, terá um sufixo .txt (exemplos: carta.txt, contrato.txt, etc.);

(2) o arquivo de texto comprimido, cujo nome, **obrigatoriamente**, terá um sufixo .xtx (exemplos: carta.xtx, contrato.xtx, etc.).

O seu programa poderá ler um arquivo normal ou comprimido, dependendo apenas do sufixo do nome do arquivo de entrada dado pelo usuário (que **obrigatoriamente** terá que ser .txt ou .xtx). Analogamente, o programa poderá gravar um arquivo normal ou comprimido, dependendo apenas do sufixo do nome do arquivo de saída dado pelo usuário (que **obrigatoriamente** terá que ser .txt ou .xtx).

Concluindo, acrescentaremos os itens (8) e (10) que ficaram faltando no esquema de funcionamento do programa que demos no início do enunciado :

(8) se o arquivo de entrada é um arquivo comprimido (extensão .xtx), a linha lida (que está no modo comprimido) deve ser convertida em uma linha normal, "expandindo" as sequências uniformes "abreviadas", antes de fazer as substituições de substrings;

(10) se o arquivo de saída é um arquivo comprimido (extensão .xtx), a linha em memória (que está no modo normal) deve ser convertida em uma linha no modo comprimido, "abreviando" as sequências uniformes de tamanho maior ou igual a 4, antes de gravá-la.

Finalmente, mais um pequeno detalhe : o seu programa deve poder ser usado não só para substituir substrings, mas também para comprimir e descomprimir arquivos, sem fazer substituições. Para isto, basta convencionar que, quando o usuário responde com o string nulo ao pedido do string a substituir (item (3) da numeração acima), ele não está querendo substituir substring, mas apenas comprimir ou descomprimir um arquivo.

IMPORTANTE: Todo exercício-programa deve seguir as observações contidas no folheto distribuído juntamente com o primeiro exercício-programa. Neste folheto estão descritas as diretrizes para forma de entrega do exercício, aspectos importantes na avaliação, etc.