Universidade de Brasília
Departamento de Ciência da Computação/IE
Semestre: 02/2013
Disciplina: Computação Básica

Trabalho Prático 3

O objetivo desta terceira etapa é que o aluno aprenda a utilizar **matrizes, alocação dinâmica de memória e utilização de variáveis globais** em algoritmos e a implementá-las em programas que executem corretamente no computador.

Especificação da terceira etapa:

Problema: ALTERAR O TRABALHO 2 NOS SEGUINTES PONTOS:

1. UTILIZAR VARIÁVEIS GLOBAIS DE FORMA ADEQUADA:
   Neste trabalho, você deve aprender a utilizar variáveis globais de forma correta. Variável global é aquela que é declarada no escopo global, ou seja, no mesmo nível em que funções são criadas. Em outras palavras, para criar uma variável global você deve declarar uma variável fora de todas as funções (inclusive da main), no início do arquivo fonte. Ou seja, uma variável global deve possuir o mesmo nível de identação de declarações de funções: ou seja, nenhum!

   A escolha de quais variáveis devem ser globais em um programa deve ser feita com cuidado. Normalmente, variáveis que contêm dados ou apontam para regiões de memória que precisam ser manipuladas por diversas funções do programa são globais. Isto não quer dizer que, por exemplo, variáveis contadoras de laços (loops) devem ser globais. Não é porque diversas funções utilizam contadores de laços, que você deve declarar um inteiro global chamado "i". Se você tentar utilizar um contador de laço global, você irá notar que o comportamento do programa será totalmente incorreto, se, por exemplo, existir um laço "para" que utiliza esta variável global para realizar a contagem e dentro deste laço há uma chamada à uma função que possui outro laço "para" que utiliza a mesma variável global "i". Após a função retornar, a variável "i" estará com o último valor do laço desta função, o que muito provavelmente irá provocar o mal funcionamento do laço que chamou a função. Outro exemplo, são variáveis utilizadas para realizar troca de valores entre variáveis. Se você utilizar um inteiro global "aux" para realizar uma troca de valores entre duas variáveis em diversas funções, muito provavelmente o programa irá funcionar de maneira incorreta. Mais um exemplo, são variáveis acumuladoras. Não é porque você precisa somar uma série de valores em diversas funções, que você deve utilizar uma variável global "total" como acumulador.

   Por outro lado, o banco de palavras e suas respectivas pontuações utilizadas em um Jogo da Forca, por exemplo, precisam estar em apenas **um** lugar da memória, durante toda a execução do programa. A informação (índice) de qual palavra está sendo adivinhada na rodada corrente, o estado atual da palavra decifrada, o estado atual do alfabeto, a pontuação total do jogador, a matriz com o desenho do personagem sendo enforcado e qual a próxima parte do corpo que deve ser desenhada são dados cujos valores irão mudar durante a execução do programa, diferentemente do banco de palavras e pontuações. No entanto, estes

dados também só precisam estar em um lugar da memória. Inclusive a pontuação total, que por acaso é uma variável de acumulação. Ou seja, sempre há exceções para considerar.

Contudo, perceba que é possível implementar um programa que utiliza apenas variáveis locais, por mais que muitas destas variáveis possam ser globais (isso foi feito no trabalho 2). A vantagem em declarar variáveis como globais, as que podem ter este privilégio, está em simplificar a interface de diversas funções. Não é necessário, por exemplo, que a string com o estado atual do alfabeto de um Jogo da Forca precise ser passada como argumento de uma função, que repassa esta mesma string como argumento de outra função e assim por diante. Todas as variáveis globais são acessíveis dentro de qualquer função. Outra vantagem, do ponto de vista de desempenho, tanto temporal quanto espacial, é que o compilador não precisa gerar instruções de máquina para empilhar argumentos de função. O computador não perde tempo colocando argumentos na pilha de chamada/registradores e não perde espaço criando cópias e mais cópias de uma mesma variável.

Concluindo, chegamos ao seguinte princípio: uma variável deve ser local, até que hajam razões suficientes para que na verdade ela seja global; e não o contrário. Ou seja, não é normal declarar inicialmente uma variável como global e somente mover sua declaração para dentro do corpo de uma função (tornando-a uma variável local) se aparecerem boas razões para isto. A mudança natural do escopo de uma variável é mudar de escopo local para escopo global, quando surgem fortes razões. O contrário pode acontecer, mas não é comum.

Neste trabalho, por razões didáticas, as variáveis que devem ser globais já estão indicadas. **Somente as seguintes variáveis DEVEM ser globais:**

```
int m;                          /* quantidade de palavras (ou linhas) */
char** palavras;                /* matriz[m][n] de palavras */
int* pontos;                    /* vetor[m] de pontos de cada palavra */
int palavra_alvo;               /* indice da palavra atual */
char* palavra_decifrada;        /* string[n] da palavra decifrada */
char alfabeto[27];              /* estado atual do alfabeto */
int pontos_total;               /* pontuacao total do jogador */
char forca[23][31];             /* matriz do desenho da forca */
int parte_corpo;                /* indica a proxima parte do corpo */
```

obs: você pode e deve continuar utilizando os nomes das variáveis que você já criou no trabalho 2. Os nomes acima são apenas exemplos de nomenclatura e declaração.

2. MATRIZ ALOCADA DINAMICAMENTE PARA ARMAZENAR AS PALAVRAS:

No trabalho 2 o arquivo texto (PALAVRAS.TXT) contendo as palavras permanecia aberto durante toda a execução do jogo. Neste trabalho 3 você deverá criar uma função que abre o arquivo texto e carrega o mesmo (por completo) para uma matriz alocada dinamicamente.

Antes de fazer a alocação dinâmica é necessário descobrir o número de palavras e tamanho da maior palavra armazenada no arquivo, pois estes dados são respectivamente o número de linhas e colunas da matriz que será alocada. Lembrando que a número de colunas é o tamanho da maior palavra mais 1 (um). Para descobrir estas informações será necessário abrir o arquivo e percorrê-lo.

Para entender o funcionamento da alocação dinâmica, primeiro consulte os slides feitos pelo monitor (e tutor) Matheus Pimenta que estão no Moodle. Naturalmente, é necessário ter pleno entendimento de ponteiros antes de iniciar a leitura dos slides. Abaixo segue uma alternativa simples para alocar dinamicamente uma matriz m x n onde m=2 e n=3.

```
int m;           /* esta variavel sera global */
char** palavras; /* esta variavel sera global */

int n, i;

m = 2; n = 3;
palavras = (char**) malloc(sizeof(char*)*m);
for (i = 0; i < m; i++)
      palavras[i] = (char*) malloc(sizeof(char)*n);
```

O procedimento acima deve ser realizado dentro de uma função que recebe uma string contendo o nome físico do arquivo texto. O ponteiro para a matriz alocada dinamicamente nesta função (`palavras`) será global (ou seja, declarado fora de todas as funções, inclusive da `main`).

Além da matriz para guardar as palavras, você também deverá alocar dinamicamente um vetor de inteiros de tamanho `m` para guardar a pontuação de cada palavra. Isto é, o valor armazenado na posição 0 (zero) do vetor corresponde a pontuação da palavra armazenada na linha 0 (zero) da matriz.

A string `palavra_decifrada` também deve ser alocada dinamicamente, pois seu tamanho é calculado como o tamanho da maior palavra do arquivo + 1 (um, para o caractere '\0').

Para realizar a primeira passagem, descobrindo tamanhos de palavras, você pode utilizar a função `fgetc`, que lê um caractere do arquivo texto. Ao se deparar com um espaço, você encontra o tamanho de uma palavra.

Para gravar as palavras lidas e pontuação respectivamente na matriz e no vetor alocados, você deverá reabrir o arquivo (ou seja, fechar e abrir novamente) iniciando a leitura do começo. Nesta segunda passagem, basta ler a palavra da mesma forma como foi feito no trabalho 2.

Também deverá ser implementada uma função para liberar a memória utilizada pela matriz, pelo vetor e pela string. Esta função deverá ser chamada no final da main. Ela deve retornar void e não recebe nada como argumento, pois os ponteiros a serem liberados e a variável `m` (que indica o número de linhas) são globais!

```
for (i = 0; i < m; i++)
      free(palavras[i]);
free(palavras);
free(pontos);
free(palavra_decifrada);
```

Uma alternativa **bônus**, que introduz duas melhorias notáveis na implementação do programa, é realizar apenas uma passagem pelo arquivo texto (e não duas), utilizando

inicialmente a função `malloc` para alocar o caractere inicial de uma palavra e então utilizar as funções `fgetc` e `realloc`, para ler um caractere do arquivo e aumentar o espaço já alocado para aquela palavra. Ao final, basta colocar o ponteiro da string com a palavra que acabou de ser lida no vetor de ponteiros (`palavras`). As duas melhorias são: percorrer o arquivo texto apenas uma vez; e alocar apenas o espaço necessário para cada palavra, alocando uma "matriz" que possui linhas de tamanho variável. Uma pontuação bônus será atribuída aos trabalhos que implementarem esta alternativa corretamente, sem deixar ocorrer nenhum vazamento de memória.

3. DESENHAR AS PARTES DO CORPO DO BONECO DA FORCA UTILIZANDO UMA MATRIZ:

   Neste trabalho, você deve desenhar as partes do corpo do personagem que está sendo enforcado. Para isto, utilize a matriz global `forca`, de 23x31 caracteres. São 30 colunas de caracteres que serão impressos e a última coluna é para os caracteres '\0'. Você deve criar **no mínimo** as seguintes funções:
   1) Função para inicializar a matriz apenas com o desenho da forca, sem o personagem e a corda. Esta função deve inicializar toda a matriz com espaços, colocando os caracteres que desenham apenas a forca. Veja os desenhos anexos no arquivo texto FORCA.TXT;
   2) Função para exibir a matriz, com um laço que imprime suas 23 strings;
   3) Função para colocar na matriz os caracteres que representam a cabeça;
   4) Função para colocar na matriz os caracteres que representam o tronco;
   5) Função para colocar na matriz os caracteres que representam o braço direito;
   6) Função para colocar na matriz os caracteres que representam o braço esquerdo;
   7) Função para colocar na matriz os caracteres que representam a perna direita;
   8) Função para colocar na matriz os caracteres que representam a perna esquerda;
   9) Função para colocar na matriz os caracteres que representam a corda da forca.

   Observação: As funções de 3 à 9 devem ser eficientes! O computador não deve perder tempo atribuindo espaços à celulas da matriz.

   A lógica de uma rodada neste trabalho deve ser a seguinte:

   A matriz da forca é inicializada e exibida.
   Na primeira vez que o jogador faz um palpite incorreto, a cabeça deve ser colocada na matriz e esta deve ser exibida.
   Se o jogador realizar mais uma jogada incorreta, colocar o tronco e exibir a matriz.
   Se o jogador realizar mais uma jogada incorreta, colocar o braço direito e exibir a matriz.
   Se o jogador realizar mais uma jogada incorreta, colocar o braço esquerdo e exibir a matriz.
   Se o jogador realizar mais uma jogada incorreta, colocar a perna direita e exibir a matriz.
   Se o jogador realizar mais uma jogada incorreta, colocar a perna esquerda e exibir a matriz.
   Se o jogador realizar mais uma jogada incorreta, colocar a corda da forca e exibir a matriz.
   Se a corda chegar a ser exibida, o jogador perde o jogo e não tem mais chances de tentar acertar a palavra.

   As dimensões e desenhos anexos são apenas sugestões. Você pode mudar estes detalhes, caso queira implementar algo mais bem feito. Você pode inclusive bolar maneiras dinâmicas, desde que sejam maneiras eficientes, de carregar desenhos do arquivo FORCA.TXT com as

alterações que achar necessárias, de modo que o usuário do programa seja capaz de determinar as dimensões da matriz e seus desenhos. É fortemente aconselhado que você tente implementar estes adicionais, pois é divertido e com certeza te ajudará no futuro!

Observações Gerais:

1. Incluir cabeçalho como comentário (ou seja, entre /* */), no programa fonte, de acordo com os critérios de avaliação dos trabalhos (Disponível no Moodle).
2. A data de entrega do programa é: 01/12/2013 (dom) até às 23:55 hs, via moodle.
3. O arquivo fonte deve ser compactado em um arquivo .zip ou .rar, seguindo as regras de nomenclatura especificadas na guia do aluno.
4. Sigam as determinações quanto às entradas e saídas do programa. Serão descontados pontos de programas que tiverem de ter a entrada e/ou a saída corrigidas. Se tiverem alguma dúvida quanto à formatação de entrada e de saída, entrem em contato com os monitores com antecedência em relação ao prazo de entrega do trabalho.