UEZO - Centro Universitário Estadual da Zona Oeste
Ciência da Computação
Tec. em Análise e Desenvolvimento de Sistemas
INF5321 - CCB1044: Estruturas de Dados I
Profa. Dilza Szwarcman

LISTA DE EXERCÍCIOS
Alocação Dinâmica

OBS: Para os exercícios abaixo, crie um projeto "Console Application", em C, no ***Code Blocks*** *e, para execução, utilize o Debugger com breakpoints e acompanhe os valores das variáveis na janela Watches. Faça impressão de tela da janela Watches e do console e inclua na resposta junto com o código fonte.*

1. Implemente uma função que receba dois vetores de inteiros v1 e v2, mais um inteiro N com o tamanho dos vetores. Sua função deve alocar e retornar um vetor de inteiros de tamanho N onde o elemento na posição i de v3 é a soma dos elementos na posição i de v1 e v2. Sua função deve ter a seguinte declaração:

   ```
   int * soma_vetores(int *v1, int *v2, int N);
   ```

   Para testar seu programa, crie uma função main() que chame sua função `soma_vetores` e imprima os valores somados.

2. Escreva um programa que aloque dinamicamente uma matriz (de inteiros) de dimensões definidas pelo usuário e preencha a matriz com números aleatórios (*rand*). Em seguida, implemente uma função que receba um valor, retorne 1 caso o valor esteja na matriz ou retorne 0 caso não esteja na matriz.

3. Faça um programa que leia dois números n e m e, usando ponteiros duplos (ponteiro para ponteiro):

   a. Crie e leia uma matriz *n x m* de inteiros;

   b. Crie e construa uma matriz transposta (trocando linhas por colunas) *m x n* de inteiros.

   Na sua função *main*(), mostre as duas matrizes.

   Dica: lembre-se de que uma matriz é um vetor de vetores.

4. Faça um programa para armazenar em memória um vetor de dados contendo 1500 valores do tipo *int*, usando a função de alocação dinâmica de memória *calloc*:

   a. Faça um loop e verifique se o vetor contém realmente os 1500 valores inicializados com zero (conte os 1500 zeros do vetor).

   b. Atribua para cada elemento do vetor o valor do seu índice junto a este vetor.

   c. Exibir na tela os endereços dos 10 primeiros e dos 10 últimos elementos do vetor.

5. Faça um programa que pergunte ao usuário quantos valores ele deseja armazenar em um vetor de *doubles*, depois use a função *malloc* para reservar (alocar) o espaço de memória de acordo com o especificado pelo usuário. Use esse vetor dinâmico como um vetor comum, atribuindo aos 10 primeiros elementos do vetor valores aleatórios (*rand*) entre 0 e 100. Exiba na tela os valores armazenados nos 10 primeiros elementos do vetor (o vetor deve ter um tamanho mínimo de 10 *doubles*).

6. Faça um programa que simule ’virtualmente’ a memória de um computador: o usuário começa especificando o tamanho da memória (define quantos bytes tem a memória), e depois ele irá ter 2 opções: inserir um dado em um determinado endereço, ou consultar o dado contido em um determinado endereço. A memória deve iniciar com todos os dados zerados.

7. Baseado no programa anterior, implemente um mecanismo para associar nomes as posições de memória (um nome de uma posição de memória tem até 10 caracteres, com o '\0'). O usuário irá poder usar 5 opções diferentes para manipular a memória:

   1) Associar um nome com uma posição de memória;

   2) Informar um endereço e um valor para armazenar neste endereço;

   3) Informar um nome de uma posição de memória e armazenar um valor nesta posição;

   4) Pedir para recuperar o dado contido em uma posição de memória;

   5) Pedir para recuperar o dado, indicando o nome da posição de memória onde ele se encontra.

8. Faça um laço de entrada de dados, onde o usuário deve digitar uma sequência de números, sem limite de quantidade de dados a ser fornecida. O usuário irá digitar os números um a um, sendo que caso ele deseje encerrar a entrada de dados, ele irá digitar o número Zero. No final, todos os dados digitados deverão ser apresentados na tela.

   Atenção, os dados devem ser armazenados na memória deste modo: faça com que o programa inicie criando um ponteiro para um bloco (vetor) de 10 valores inteiros, e alocando dinamicamente espaço em memória para este bloco; depois, caso o vetor alocado fique cheio, aloque um novo vetor do tamanho do vetor anterior adicionado com espaço para mais 10 valores (tamanho N+10, onde N inicia com 10); repita este procedimento de expandir dinamicamente com mais 10 valores o vetor alocado cada vez que o mesmo estiver cheio. Assim o vetor irá ser ’expandido’ de 10 em 10 valores.

9. Considere um cadastro de produtos de um estoque, com as seguintes informações para cada produto:

   - Código de identificação do produto: representado por um valor inteiro
   - Nome do produto: com ate 50 caracteres
   - Quantidade disponível no estoque: representado por um número inteiro

- Preço de venda: representado por um valor real

a. Defina uma estrutura, denominada *produto*, que tenha os campos apropriados para guardar as informações de um *produto*

b. Crie um conjunto de *n* produtos (*n* é um valor fornecido pelo usuário) e peça ao usuário para entrar com as informações de cada produto

c. Encontre o produto com o maior preço de venda e imprima na tela

d. Encontre o produto com a maior quantidade disponível no estoque e imprima na tela.

10. Faça um programa que:

a. Crie uma matriz de distancias entre n cidades diferentes;

b. Peça para o usuário entrar com as distâncias entre as cidades;

c. Exiba na tela a matriz de distancias criada;

Quando o usuário digitar o número de duas cidades o programa deverá retornar a distância entre elas.

11. O código abaixo implementa parte de uma agenda de telefones bem simples. Cada registro da agenda (contato) é composto apenas de nome e telefone. O usuário poderá inserir, remover ou listar os contatos que serão mantidos em um vetor de ponteiros de contatos, alocado dinamicamente. Pede-se:

a. Implementar a função inic_agenda() que inicializa a agenda, alocando um espaço inicial em memória para BLOCK ponteiros de contatos.

b. Implementar a função insere() que insere um contato fornecido pelo usuário. Lembrar que tal função deve verificar se o espaço de memória alocado para os ponteiros de contatos será esgotado com a inserção do novo contato, devendo reajusta-lo, se necessário, em incrementos de tamanho BLOCK.

c. Fazer o rastreio da função deleta(), mostrando o que ocorre se o usuário optar por deletar o terceiro registro de uma agenda com os cinco registros abaixo:

Maria	2222-2222
José	3333-3333
Danilo	4444-4444
Carla	5555-5555
João	6666-6666

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #define MAX 3      //numero máximo de BLOCKs que podem ser alocados
4.  #define BLOCK 5

5.  typedef struct _contato
```

```
{
    char nome[30];
    char tel[8];
} contato;

contato **agenda;
int num_contatos = 0;
int num_blocos = 0;

void inic_agenda(void);
void insere(void);
void deleta(void);
void lista(void);
int item_menu(void);

int main(void)
{
    int item;
    inic_agenda();
    for(;;)
    {
        item = item_menu();
        switch(item)
        {
        case 1:
            insere();
            break;
        case 2:
            deleta();
            break;
        case 3:
            lista();
            break;
        case 4:
            exit(0);
        }
    }
}

/* Retorna item de menu selecionado*/
int item_menu(void)
{
    char s[80];
    int c;

    printf("\n");
    printf("1. Inserir um contato\n");
    printf("2. Excluir um contato\n");
    printf("3. Listar contatos\n");
    printf("4. Sair\n");

    do
    {
        printf("\nEntre com sua escolha: ");
        gets(s);
        c = atoi(s);
    }
    while(c<0 || c>4);

    return c;
}
```

```
66.
67.  /* Apaga um contato */
68.  void deleta(void)
69.  {
70.      int indice;
71.     char s[10];
72.
73.     if (num_contatos ==0)
74.     {
75.         printf("\nAgenda vazia\n");
76.         return;
77.     }
78.     printf("\nEntre com o no. do contato: ");
79.     gets(s);
80.     indice = atoi(s);
81.     free (agenda[indice-1]);
82.     agenda[indice-1] = agenda[num_contatos -1];
83.     num_contatos--;
84.
85.     if (num_contatos <(num_blocos * BLOCK))
86.     {
87.          agenda =(contato **)realloc (agenda,(num_blocos - 1)*
88.              sizeof(contato *));
89.     }
90. }
91.
92. /* Mostra a lista de contatos na tela. */
93. void lista(void)
94. {
95.     int t;
96.     if (num_contatos ==0)
97.     {
98.         printf("\nAgenda vazia\n");
99.         return;
100.     }
101.     for(t=0; t<num_contatos; ++t)
102.     {
103.         printf("(%d) %s   %s\n", t+1, agenda[t]->nome, agenda[t]->tel);
104.     }
105.     printf("\n\n");
106. }
```