Lista 9 de Exercícios

Exercícios sobre Estruturas

1. Uma empresa que organiza concursos está realizando um grande processo seletivo e conta com um sistema informatizado para auxiliar no gerenciamento de informações sobre os candidatos. Nesse sistema, as informações referentes aos candidatos são armazenadas em um vetor de ponteiros para dados estruturados do tipo *Candidato*, conforme descrito a seguir:

```
typedef struct candidato {
     int inscr;       /*  numero de inscricao */
     char nome[81];   /*  nome do candidato   */

     Data nasc;       /*  data de nascimento  */
     Local *loc       /*  local de prova      */
     Notas nt;        /*  notas de prova      */
} Candidato;
```

O campo *nasc* refere-se à data de nascimento do candidato e é do tipo *Data*, que corresponde ao tipo estruturado descrito abaixo.

```
typedef struct data {
     int dia, mes, ano;
} Data;
```

O campo *loc* refere-se às informações sobre o local de realização das provas e é um ponteiro para o tipo *Local*, que corresponde ao tipo estruturado descrito abaixo.

```
typedef struct local {
     char ender[81];   /*  endereço do local de provas */
     int sala;         /*  numero            sala      */
} Local;
```

O campo *nt* refere-se às informações sobre as notas de prova do candidato e é do tipo *Notas*, que corresponde ao tipo estruturado descrito abaixo.

```
typedef struct notas {
     float geral;  /* prova de conhecimentos gerais */
     float especifica; /* prova de conhecimentos
especificos */
} Notas;
```

a. Faça um programa que leia as informações de n candidatos. Crie uma variável alocada dinamicamente que armazene as informações lidas.

b. Faça um menu de opções com a opção ‘1’ para leitura dos dados de candidatos e a opção ‘2’ para impressão de todas as informações dos candidatos.

c. Faça com que uma opção do menu permita alterar o endereço e a sala do local de provas de um determinado candidado.

2. Tendo o seguinte código:

```
typedef struct data {
   int dia;
   int mes;
   char nomeMes[12];
} Data;

main()
{
    Data        aniversarios[3]        =        {        {5,1,"JANEIRO"},
{4,2,"FEVEREIRO"},{10,3,"MAIO"} };
    int a;
    Data *p_dt;
    p_dt=&aniversarios[2];
    printf("Nome do mês %d é: %s.", p_dt->mes, p_dt->nomeMes);
    strcpy(p_dt->nomeMes, "MARÇO");
    printf("\nNúmero de letras : %d=", strlen(p_dt->nomeMes));
    Data p_dt2 = (Data *) malloc(sizeof(Data));
    p_dt2 = &aniversarios[0];
}
```

a. O programa acima pode ter no máximo 1 erro. Caso tiver, explique qual é o erro.

b. Qual o valor mostrado na tela no segundo (último) printf?

3. Uma instituição de pesquisa recolheu amostras de estados brasileiros a respeito do salário da população. Tendo o seguinte código:

```
char * estado[27] =
{"AC","AL","AM","AP","BA","CE","DF","ES","GO","MA","MG","MS","MT","PA",
"PB","PE","PI","PR","RJ","RN","RO","RR","RS","SC","SE","SP","TO"};

typedef struct assalariados {
    char nome[51];
    char sexo;
    int idade;
    float salario;
    char estado[3];
} Assalariados;

Assalariados * cadastra(char *nome, char sexo, int idade, float
salario, char *estado);
```

```
void relatorio(Assalariados ** ptr, int numAssalariados);
void imprime(Assalariados ** ptr, int numAssalariados);

main(void) {
    Assalariados **pessoas;
    int numAssalariados=3;
    pessoas = (Assalariados **) malloc (numAssalariados *
sizeof(Assalariados *));
    pessoas[0] = cadastra("Fulano de tal", 'M',45,1500.00, "RJ");
    pessoas[1] = cadastra("Ciclano", 'M',50,2500.00, "RS");
    pessoas[2] = cadastra("Beltrano", 'M',42,500.00, "RS");
    imprime(pessoas, numAssalariados);
    relatorio(pessoas,numAssalariados);
}
```

a. Implemente a função *cadastra*, que atribui os parâmetros recebidos aos campos da estrutura Assalariados.

b. Implemente a função *imprime*, que imprime todos os elementos do vetor de Assalariados.

c. Implemente a função *relatório*, que imprime a quantidade de homens por estado com mais de 40 anos que recebem salários superiores a R$ 1000,00. Caso a quantidade no estado analisado seja zero, então não imprime nada. A mensagem de impressão deve ser:

```
Estado=%s com %d homens maiores de 40 anos e salário maior que R$ 1000,00
```

4. Tendo o seguinte trecho de código (compilável e executável corretamente):

```
#define MAX 4
#define MAXNOME 31
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void atribui(char **nomes, int indice, char *nome);
char * get_sobrenome(char *nome);
main()
{
    int i;
    char **nomes;
    char *sobrenome;
    nomes = (char **) malloc(MAX*sizeof(char *));
    for ( i=0 ; i<MAX ; i++) {
        nomes[i] = (char *) malloc(sizeof(char)*MAXNOME);
    }
    atribui(nomes,0,"Fulano Silva"); atribui(nomes,1,"Maria do Carmo");
    atribui(nomes,2,"Beltrano  Belmonte");  atribui(nomes,3,"Pedro  dos
Santos");
    for (i=0 ; i<MAX ; i++) {
        sobrenome = get_sobrenome(nomes[i]);
        printf("\n%s ", sobrenome);
        printf(" %d",strlen(sobrenome)>5?i:0);    // 2° printf
    }
```

```
    printf("\n%c", nomes[0][3]); // 3° printf
}
```

a. Implemente a função atribui, conforme o seu protótipo (dica: somente 1 linha de código no corpo da função).

b. Implemente a função get_sobrenome, conforme o seu protótipo, que retorna a última palavra do nome.

c. O que será impresso no segundo *printf*?

d. O que será impresso no terceiro *printf*?

5. Considere um sistema de apoio escolar que armazena as informações sobre as notas dos alunos de uma turma em um vetor de ponteiros para variáveis do tipo Aluno, descrito a seguir:

```
typedef struct endereco {
   char rua[100];       /*  Nome da rua           */
   int numero;          /*  Numero do imovel      */
} Endereco;

typedef struct notas {
   float p1, p2, p3;    /*  Notas nas provas      */
} Notas;

typedef struct aluno {
   int mat;             /*  Matricula do aluno    */
   char nome[81];       /*  Nome do aluno         */
   Notas nota;          /*  Notas nas provas      */
   Endereco *end;       /*  Endereco do aluno     */
} Aluno;

main(){
    Aluno **alunos;
    alunos = alocaAlunos(3);
    atribui(alunos,0,10,"Um",1,1,1,"Getulio Vargas",100);
    atribui(alunos,1,20,"Dois",8,8,8,"Amaral Peixoto",200);
    atribui(alunos,2,30,"Tres",9,9,9,"Ouro Verde",300);
}
```

5a. Escreva uma função que aloca espaço em memória para `n` alunos. A função tem o seguinte protótipo: `Aluno ** alocaAlunos(int n);`

5b. Escreva uma função que atribui a um determinado elemento do vetor de ponteiros para estruturas os dados passados como parâmetro. A função recebe o vetor a, o índice do vetor e os dados a serem preenchidos na estrutura. A função tem o seguinte protótipo:

```
void atribui(Aluno ** a, int indice, int matricula, char *nome, float
nota1, float nota2, float nota3, char *nomeRua, int numero);
```

6. Um cadastro de pessoas é representado por um vetor de ponteiros para o tipo Pessoa, conforme descrito a seguir:

```
struct pessoa {
   int codigo;
   char nome[81];
};
typedef struct pessoa Pessoa;
```

Considere que esse vetor está em ordem crescente de código, e, além disso, o código é a identificação única da pessoa. Aplicando a técnica de busca binária, implemente uma função que verifique se um código fornecido como parâmetro existe no vetor. A função retorna o ponteiro para esta pessoa (Pessoa *). Caso contrário, a função deve retornar NULL. A função recebe como parâmetros o ponteiro vet, para o primeiro elemento do vetor, o inteiro n, que representa o tamanho do vetor, e o código. Seu protótipo é:

```
Pessoa* busca(int n, Pessoa** vet, int codigo);
```

Faça também um programa para testar essa função.

7. Considere o tipo Aluno, que representa um aluno da disciplina, e o tipo Prova, que representa a prova de um aluno, ambos descritos a seguir:

```
struct aluno {
   int mat;                /*  Matricula do aluno        */
   char nome[81];          /*  Nome do aluno             */
};
typedef struct aluno Aluno;

struct prova {
   Aluno a;                /*  Aluno que fez a prova     */
   float q1, q2, q3, q4;   /*  Nota em cada questao      */
};
typedef struct prova Prova;
```

O seguinte algoritmo de ordenação bolha foi implementado, utilizando as funções auxiliares compara e troca, para ordenar um vetor de ponteiros para o tipo Prova:

```
   void ordena(int n, Prova** v)
   {
         int fim,i;
         for(fim=n-1; fim>0; fim--)
               for(i=0; i<fim; i++)
                     if(compara(v[i],v[i+1]))
                           troca(&v[i],&v[i+1]);
   }
```

Essa função ordena o vetor em ordem decrescente de nota da prova --- que equivale à soma das notas das quatro questões ---, com desempate pela ordem alfabética do nome do aluno. Por

exemplo, se Maria e Ana tiram 6.0 e Sandra tira 7.0, a ordem deve ser Sandra, Ana e Maria. Implemente as funções compara e troca de forma que o algoritmo descrito funcione corretamente.

8. Uma agenda é representada por um vetor de ponteiros para o tipo Compromisso, que representa as informações sobre um compromisso diário, conforme descrito a seguir:

```
struct data {
   int dd, mm, aa;        /*  Dia, mes e ano              */
};
typedef struct data Data;

struct compromisso {
   char descricao[81];    /*  Descricao do compromisso    */
   Data dta;              /*  Data do compromisso         */
};
typedef struct compromisso Compromisso;
```

Considere que esse vetor está em ordem cronológica, ou seja, ordem crescente de acordo com a data de cada compromisso, e, além disso, não há mais de um compromisso com uma mesma data. Aplicando a técnica de busca binária, implemente uma função que, dada uma data (dia, mês e ano), se há no vetor algum compromisso com essa data, retorna o ponteiro para este compromisso. Caso contrário, a função deve retornar NULL. A função recebe como parâmetros o ponteiro vet, para o primeiro elemento do vetor, o inteiro n, que representa o tamanho do vetor, e os inteiros d, m e a, representando uma data. Seu protótipo é

```
Compromisso* busca(int n, Compromisso** vet, int d, int m, int a);
```

9. Faça a função *main* para funcionar com a função *ordena*.

```
void ordena (ALUNO turma[], int tam) {
    int i, foraOrdem , jaOrdenados = 0;
    ALUNO temp;
    do {
        foraOrdem = 0;
        for (i = 0; i < 4 - 1 - jaOrdenados ; i++) {
            if (turma[i]. media > turma[i+1]. media) {
                temp = turma[i];
                turma[i] = turma[i+1];
                turma[i+1] = temp ;
                foraOrdem = 1;
            }
        }
        jaOrdenados ++;
    } while ( foraOrdem );
}
```

10. Faça as funções que são invocadas pela função *main*.

```
int main (void) {
    struct aluno turma[MAX];
    le( turma );
```

```
      puts (" Imprimindo dados lidos da turma.");
      puts (" Digite qualquer coisa para continuar .");
      getchar ();
      imprime ( turma );
      ordena_medias ( turma );
      puts (" Imprimindo dados ordenados da turma.");
      puts (" Digite qualquer coisa para continuar .");
      getchar ();
      imprime ( turma );
      getchar ();
}
```

11. Criar um tipo Aluno e um tipo Materia tal como dado abaixo:

```
typedef struct Aluno {
    int matricula;
    float *vNotas; // Armazena as 5 notas de um aluno ao longo de um ano.
    char nome[100];
}Aluno;

typedef struct Matéria {
    Aluno *V; // Armazena a informação de n alunos !!
    float media[5]; // Armazena as 5 médias do ano.
    int nAlunos // Número de alunos matriculados no curso.
}Materia;
```

(A) Criar uma função *Aluno* fillAluno()* que preenche os campos da estrutura do tipo Aluno com dados fornecidos pelo usuário.
(B) Criar uma função *Materia * fillMateria(int numAlunos)* que preenche os campos da estrutura do tipo Materia realizando chamadas a função fillAluno em um número de vezes igual ao número contido na variável numAlunos.
(C) Criar uma função *void mediaMateria(Materia *m1)* que fornece a média de cada prova do ano para os alunos contidos na variável m1 e assim preenche o campo media de m1.
(D) Criar uma função *void mostraMateria(Materia *m1)* que mostra as informações dos alunos contidas na variável m1.
(E) Criar um programa que ilustra o funcionamento das funções anteriores.

12. Criar um tipo Livro e um tipo Biblioteca tal como dado abaixo:

```
typedef struct Livro {
    int ano;
    char titulo[100];
    char autor[100];
    int nVolume; // Número de exemplares de um dado livro.
    float preco;
}Livro;

typedef struct Biblioteca {
    Livro **V; // Armazena a informação de n livros !!
    int nLivros // Número de livros existentes na biblioteca.
}Biblioteca;
```

(A) Criar uma função *Livro * fillLivro()* que preenche os campos da estrutura Livro e retorna esse novo livro.
(B) Criar uma função Biblioteca * fillBiblioteca(int numLivros) que preenche os campos da estrutura do tipo Biblioteca realizando chamadas a função fillLivro em um número de vezes igual ao número contido na variável numLivros.
(C) Criar uma função void valorBiblioteca(Biblioteca * b1, int numLivros) que fornece o montante gasto para se comprar todos os exemplares existentes na biblioteca.
(D) Criar uma função Livro * maiorBiblioteca(Biblioteca * b1, int numLivros) que retorna o livro com maior número de exemplares.
(E) Criar um programa que ilustra o funcionamento das funções anteriores.