

Lista de Exercícios 2 – assuntos: ESTRUTURAS E PONTEIROS

LISTA PARA FIXAR O CONTEÚDO E ESTUDAR PARA A 2ª AVALIAÇÃO

NÃO PONTUA, MAS SERVE COMO RECURSO CASO VOCÊ PRECISE DE ATÉ 0,5 PONTO

Instruções para a resolução e entrega da lista.

1 – A entrega da lista é individual, entretanto incentivo o estudo em grupo, para compartilhamento de experiências e conhecimentos.

2 – As respostas devem ser dadas logo abaixo de cada questão, no próprio arquivo da lista.

3 – PRAZO – VEJA NA DESCRIÇÃO DA TAREFA NO BBOARD.

1. Quais são os valores mostrados pelo programa em C abaixo?

```
1 int main(void){  
2     int i = 99, j;  
3     int *p;  
4  
5     p = &i;  
6     j = *p + 100;  
7     printf("i = %d, j = %d, *p = %d", i, j, *p);  
8 }
```

RESPOSTA I = 99, j= 199, *p = 99

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

2. Quais são os valores mostrados pelo programa em C abaixo?

```
60 int main() {
61     int    a = 5, b = 12, c;
62     int *   p;
63     int *   q;
64
65     p = &a;
66     q = &b;
67
68     printf("p = %x\tq = %x\n", p, q);
69
70     c = *p + *q;
71
72     printf("c = %d, a = %d, b = %d", c, a, b);
73 }
```

RESPOSTA - P = 61ff10, Q = 61ff0c, C = 17, A = 5, B = 12

3. Quais são os valores mostrados pelo programa em C abaixo?

```
20 int main() {
21     int    a = 4, b = 3;
22     int *   p1, * p2;
23
24     p1 = &a;
25     p2 = p1;
26
27     printf("p1 = %x\tp2 = %x\n", p1, p2);
28
29     *p2 = *p1 + 3;
30     b = b * (*p1);
31     (*p2)++;
32     p1 = &b;
33
34     printf("%d\t%d\n", *p1, *p2);
35     printf("%d\t%d\n", a, b);
36 }
```

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

RESPOSTA - p1 = 61ff14, p2 = 61ff10 * p1 = 29, *p2 = 29a = 4, b = 29

4. Quais são os valores mostrados pelo programa em C abaixo?

```
36 void func(int * px, int * py){
37
38     printf("*px = %d\t*py = %d\n", *px, *py);
39     printf("px = %x\tpy = %x\n", px, py);
40     printf("&px = %x\t&py = %x\n", &px, &py);
41
42     px = py;
43     *py = (*py) * (*px);
44     *px = *px + 2;
45 }
46
47 int main() {
48     int x, y;
49
50     scanf("%d",&x); // entrar com o valor 3
51     scanf("%d",&y); // entrar com o valor 4
52
53     func(&x, &y);
54
55     printf("x = %d, y = %d", x, y);
56 }
```

RESPOSTA - *px = 3, *py = 4

px = 61ff1c, py = 61ff18

&px = 61ff00, &py = 61ff04

x = 3, y = 18

5. A memória do computador funciona, basicamente, armazenando sequências de bytes, e cada sequência tem seu próprio tipo e endereço de memória. A quantidade de bytes de uma sequência é determinada pelo tipo, que pode ser: tipo primitivo, ponteiro, estrutura de dados ou a combinação ilimitada destes. O endereço de memória é representado

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

por um número hexadecimal na forma 0x1234 (os valores apresentados nesta questão são imaginários e não representam endereços reais válidos).

Com base nestas informações, avalie as afirmações a seguir.

- I. Um ponteiro é um tipo de variável que pode ser manipulado com os operadores "&" e "*".
- II. O que um ponteiro carrega dentro dele é um número hexadecimal, que é um endereço de memória de uma sequência de bytes de um determinado tipo.
- III. É possível realizar operações aritméticas (soma, subtração, multiplicação e divisão) sobre ponteiros.
- IV. Ao somarmos uma unidade a um ponteiro, este terá seu conteúdo somado da quantidade de bytes correspondente ao tipo da área para a qual ele aponta.

É correto o que se afirma em

A) I e IV, apenas.

B) III, apenas.

C) I e II, apenas.

D) II e III, apenas.

E) I, II e IV.

6. Observe o código abaixo e responda a pergunta.

```
1 void funcaoA (int ** pt_ptl, int * ptLista, int Lista) {  
2     // faz qualquer coisa...  
3 }  
4  
5 int main ()  
6 {  
7     int lst;  
8  
9     int * pl = (int *) malloc (sizeof(int));  
10  
11     lst = *pl;  
12  
13     funcaoA (&pl, pl, lst);  
14 }
```

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

Considere que:

- na linha 9 o malloc retorna o endereço de memória 0x456;
- que a variável pl está alocada no endereço de memória 0x777;
- no main(), a função funcaoA está sendo chamada com passagem de parâmetros.

Avalie as alternativas abaixo e escolha a única correta.

a)

no main()

- **lst representa uma variável inteira**
- **pl carrega 0x456**

na funcaoA()

- **pt_ptL carrega 0x777**
- **ptLista carrega 0x456**
- **Lista é uma cópia de lst**

b)

no main()

- lst representa uma estrutura
- pl carrega 0x777

na funcaoA()

- pt_ptL carrega 0x456
- ptLista carrega 0x456
- Lista é uma cópia de lst

c)

no main()

- lst carrega 0x456
- pl carrega 0x456

na funcaoA()

- pt_ptL carrega 0x777
- ptLista carrega 0x456
- Lista não é uma cópia de lst

d)

no main()

- lst é uma estrutura alocada em 0x456
- pl carrega 0x456

na funcaoA()

- pt_ptL carrega 0x777
- ptLista carrega 0x456
- Lista não é uma cópia de lst

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

e)

no main()

- Ist é um ponteiro para a estrutura alocada em 0x456
- pl é a estrutura alocada em 0x456

na funcaoA()

- pt_ptL carrega 0x456
- ptLista carrega 0x777
- Lista é uma cópia de Ist

7. A memória do computador funciona, basicamente, armazenando sequências de bytes, e cada sequência tem seu próprio tipo e endereço de memória. A quantidade de bytes de uma sequência é determinada pelo tipo, que pode ser: tipo primitivo, ponteiro, estrutura de dados ou a combinação ilimitada destes. O endereço de memória é representado por um número hexadecimal na forma 0x1234.

Com base nestas informações, avalie as afirmações a seguir.

- I. Para obter o endereço de memória de uma variável qualquer, utiliza-se o operador "&" antes do nome dessa variável.
- II. O que um ponteiro carrega dentro dele é sempre um número hexadecimal, independentemente do tipo de dado para o qual ele aponta.
- III. Para obter o valor que está no endereço de memória que está dentro de uma variável do tipo ponteiro, utiliza-se o operador "*" antes do nome da variável ponteiro.
- IV. Para obter o conteúdo de uma parte de uma estrutura de dados, referenciada por uma variável do tipo desta estrutura, utiliza-se o operador "." (ponto) depois do nome da variável, seguido do nome da variável da parte desejada.

É correto apenas o que se afirma em

- A) I, II e IV.
B) I, II, III e IV.
C) II e III.
D) I, III e IV.
E) I e IV.

8. Este código foi executado e produziu a saída da console.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void funcaoA (int ** pt_ptL, int * ptLista, int L) {
5
6      printf("&pt_ptL = %x\npt_ptL = %x\n*pt_ptL = %x\n**pt_ptL = %d\n",
7          &pt_ptL, pt_ptL, *pt_ptL, **pt_ptL);
8  }
9
10 int main ()
11 {
12     int l = 77777;
13
14     int * pl = (int *) malloc (sizeof(int));
15
16     *pl = l;
17
18     funcaoA (&pl, pl, l);
19 }

```

CONSOLE:

```

&pt_ptL = 62fdf0
pt_ptL = 62fe10
*pt_ptL = 151480
**pt_ptL = 77777

-----
Process exited after 0.0986 seconds with return value 0
Pressione qualquer tecla para continuar. . . _

```

Preencha a tabela abaixo com os valores correspondentes às variáveis.

	l	pl	pt_ptL	pt_Lista	L
Tipo	int	int*	int**	int*	int
Endereço	15147c	62fe10	62fdf0	62fe14	62fdec
Valor	77777	151480	62fe10	1	1

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

9. Modifique o código abaixo para coletar do teclado a quantidade de números a serem digitados e armazenados, faça a alocação dinâmica do array com a dimensão informada.

```
7  #include    <stdio.h>
8  #include    <locale.h>
9
10 int main()
11 {
12     setlocale(0, "Portuguese");
13
14     int      nros[5], i;
15
16     // entrada - capturar um número e guardar em um array, repetindo 5 vezes
17     for (i = 0; i < 5; i++) {
18         printf("Digite um número: ");
19         scanf("%d", &nros[i]);
20     }
21
22     // saída - escrever número por número, todos lidos na entrada
23     // na ordem crescente dos índices -> de 0 até 4
24     for (i = 0; i < 5; i++)
25         printf("%d\t", nros[i]);
26
27     printf("\n");
28
29     // na ordem decrescente dos índices -> 4 até 0
30     for (i = 4; i >= 0; i--)
31         printf("%d\t", nros[i]);
32 }
```

```
#include <stdio.h>
```

```
#include <stdlib.h> // Para malloc e free
```

```
int main() {
```

```
    int *nrs, i, tamanho;
```

```
    // Solicita ao usuário o tamanho do array
```

```
    printf("Digite a quantidade de números a serem armazenados: ");
```

```
    scanf("%d", &tamanho);
```

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

```
// Aloca dinamicamente o array
nrs = (int *)malloc(tamanho * sizeof(int));
if (nrs == NULL) { // Verifica se a alocação foi bem-sucedida
    printf("Erro ao alocar memória.\n");
    return 1;
}

// Entrada dos números
for (i = 0; i < tamanho; i++) {
    printf("Digite o número %d: ", i + 1);
    scanf("%d", &nrs[i]);
}

// Exibe os números na ordem de entrada
printf("\nNúmeros na ordem de entrada:\n");
for (i = 0; i < tamanho; i++) {
    printf("%d\t", nrs[i]);
}
printf("\n");

// Exibe os números na ordem inversa
printf("Números na ordem inversa:\n");
for (i = tamanho - 1; i >= 0; i--) {
    printf("%d\t", nrs[i]);
}
printf("\n");

// Libera a memória alocada
```

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

```
free(nrs);  
  
return 0;  
}
```

10. Escreva um programa em C, com alocação de memória totalmente dinâmica, para armazenar *strings* em uma matriz. Você deve capturar do teclado o tamanho da maior *string* a ser armazenada e as dimensões da matriz. Por exemplo, a maior *string* terá 124 caracteres e a matriz será de 4x5.

```
#include <stdio.h>  
  
#include <stdlib.h>  
  
#include <string.h>  
  
int main() {  
  
    int linhas, colunas, tamanhoString, i, j;  
  
    char ***matriz; // Ponteiro triplo para armazenar as strings na matriz  
  
    // Entrada das dimensões e do tamanho máximo da string  
    printf("Digite o tamanho máximo de cada string: ");  
    scanf("%d", &tamanhoString);  
  
    printf("Digite o número de linhas da matriz: ");  
    scanf("%d", &linhas);  
  
    printf("Digite o número de colunas da matriz: ");
```

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

```
scanf("%d", &colunas);

// Alocação dinâmica para a matriz
matriz = (char ***)malloc(linhas * sizeof(char **));
if (matriz == NULL) {
    printf("Erro ao alocar memória para as linhas.\n");
    return 1;
}

for (i = 0; i < linhas; i++) {
    matriz[i] = (char **)malloc(colunas * sizeof(char *));
    if (matriz[i] == NULL) {
        printf("Erro ao alocar memória para as colunas.\n");
        return 1;
    }
    for (j = 0; j < colunas; j++) {
        matriz[i][j] = (char *)malloc((tamanhoString + 1) * sizeof(char)); // +1 para o caractere
nulo
        if (matriz[i][j] == NULL) {
            printf("Erro ao alocar memória para as strings.\n");
            return 1;
        }
    }
}

// Entrada das strings
printf("Digite as strings para a matriz:\n");
for (i = 0; i < linhas; i++) {
    for (j = 0; j < colunas; j++) {
```

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

```
printf("Elemento [%d][%d]: ", i, j);

scanf(" ");

fgets(matriz[i][j], tamanhoString + 1, stdin);

matriz[i][j][strcspn(matriz[i][j], "\n")] = '\0'; // Remove o caractere de nova linha
}

}

// Exibição da matriz

printf("\nMatriz de strings:\n");

for (i = 0; i < linhas; i++) {
    for (j = 0; j < colunas; j++) {
        printf("%s\t", matriz[i][j]);
    }

    printf("\n");
}

// Liberação da memória

for (i = 0; i < linhas; i++) {
    for (j = 0; j < colunas; j++) {
        free(matriz[i][j]);
    }

    free(matriz[i]);
}

free(matriz);

return 0;
}
```

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

11. Escreva um programa em C, totalmente dinâmico, para realizar as quatro operações (CRUD) de um cadastro de alunos: **Create**, **Read**, **Update** e **Delete**. Cada operação deverá estar em uma função. Deverá ter uma função para mostra o cadastro inteiro. O cadastro deverá armazenar **nome completo, RGM e curso**. O nome deverá ser coletado do teclado e armazenado em uma área estática temporária de tamanho 1024 caracteres. Após ter o nome, alocar a área exata que irá guardar o nome. Os cursos, listados abaixo, deverão estar em um ENUMERATION, sendo que o valor armazenado será o número inteiro, mas na hora de mostrar, deverá mostrar o nome do curso.

- 1 - Análise e Desenvolvimento de Sistemas
- 2 - Ciência da Computação
- 3 - Ciência de Dados
- 4 - Gestão da Tecnologia da Informação
- 5 - Redes de Computadores
- 6 - Sistemas Para Internet

RESPOSTA

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
// Definição do ENUM para os cursos
```

```
typedef enum {
```

```
    ADS = 1,
```

```
    CC,
```

```
    CD,
```

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

```
GTI,  
RC,  
SI  
} Curso;  
  
// Estrutura para armazenar os dados de cada aluno  
typedef struct {  
    char *nome; // Nome completo (alocado dinamicamente)  
    int rgm; // RGM  
    Curso curso; // Curso (ENUM)  
} Aluno;  
  
// Variáveis globais para o cadastro de alunos  
Aluno *cadastro = NULL;  
int totalAlunos = 0;  
  
// Funções auxiliares  
void mostrarCursos() {  
    printf("Cursos disponíveis:\n");  
    printf("1 - Análise e Desenvolvimento de Sistemas\n");  
    printf("2 - Ciência da Computação\n");  
    printf("3 - Ciência de Dados\n");  
    printf("4 - Gestão da Tecnologia da Informação\n");  
    printf("5 - Redes de Computadores\n");  
    printf("6 - Sistemas Para Internet\n");  
}  
  
const char *obterNomeCurso(Curso curso) {
```

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

```
switch (curso) {  
    case ADS: return "Análise e Desenvolvimento de Sistemas";  
    case CC: return "Ciência da Computação";  
    case CD: return "Ciência de Dados";  
    case GTI: return "Gestão da Tecnologia da Informação";  
    case RC: return "Redes de Computadores";  
    case SI: return "Sistemas Para Internet";  
    default: return "Curso desconhecido";  
}  
}
```

```
// Função para criar um novo aluno
```

```
void createAluno() {  
    char buffer[1024];  
    int rgm, cursoEscolhido;  
  
    printf("\n--- Cadastro de Novo Aluno ---\n");  
  
    // Coleta do nome  
    printf("Digite o nome completo do aluno: ");  
    scanf("%[^\n]", buffer);  
  
    // Coleta do RGM  
    printf("Digite o RGM do aluno: ");  
    scanf("%d", &rgm);  
  
    // Exibição dos cursos e coleta do curso escolhido  
    mostrarCursos();
```

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

```
printf("Escolha o curso pelo número correspondente: ");
scanf("%d", &cursoEscolhido);

// Aloca mais espaço para o novo aluno
cadastro = (Aluno *)realloc(cadastro, (totalAlunos + 1) * sizeof(Aluno));
if (cadastro == NULL) {
    printf("Erro ao alocar memória.\n");
    return;
}

// Armazena os dados do novo aluno
cadastro[totalAlunos].nome = (char *)malloc((strlen(buffer) + 1) * sizeof(char));
strcpy(cadastro[totalAlunos].nome, buffer);
cadastro[totalAlunos].rgm = rgm;
cadastro[totalAlunos].curso = cursoEscolhido;

totalAlunos++;
printf("Aluno cadastrado com sucesso!\n");
}

// Função para listar todos os alunos
void readAlunos() {
    printf("\n--- Lista de Alunos ---\n");
    if (totalAlunos == 0) {
        printf("Nenhum aluno cadastrado.\n");
        return;
    }
    for (int i = 0; i < totalAlunos; i++) {
```

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

```
printf("Aluno %d:\n", i + 1);
printf(" Nome: %s\n", cadastro[i].nome);
printf(" RGM: %d\n", cadastro[i].rgm);
printf(" Curso: %s\n", obterNomeCurso(cadastro[i].curso));
}
}

// Função para atualizar os dados de um aluno
void updateAluno() {
    int rgm, cursoEscolhido;
    char buffer[1024];

    printf("\n--- Atualizar Aluno ---\n");
    printf("Digite o RGM do aluno a ser atualizado: ");
    scanf("%d", &rgm);

    for (int i = 0; i < totalAlunos; i++) {
        if (cadastro[i].rgm == rgm) {
            printf("Aluno encontrado: %s\n", cadastro[i].nome);

            // Atualiza o nome
            printf("Digite o novo nome do aluno: ");
            scanf("%[^\\n]", buffer);
            free(cadastro[i].nome); // Libera a memória do nome antigo
            cadastro[i].nome = (char *)malloc((strlen(buffer) + 1) * sizeof(char));
            strcpy(cadastro[i].nome, buffer);

            // Atualiza o curso
```

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

```
    mostrarCursos();

    printf("Escolha o novo curso pelo número correspondente: ");

    scanf("%d", &cursoEscolhido);

    cadastro[i].curso = cursoEscolhido;


    printf("Dados do aluno atualizados com sucesso!\n");

    return;
}

}

printf("Aluno com RGM %d não encontrado.\n", rgm);
}

// Função para excluir um aluno
void deleteAluno() {
    int rgm;

    printf("\n--- Excluir Aluno ---\n");

    printf("Digite o RGM do aluno a ser excluído: ");

    scanf("%d", &rgm);

    for (int i = 0; i < totalAlunos; i++) {
        if (cadastro[i].rgm == rgm) {
            // Libera a memória do nome do aluno
            free(cadastro[i].nome);

            // Remove o aluno do cadastro
            for (int j = i; j < totalAlunos - 1; j++) {
                cadastro[j] = cadastro[j + 1];
```

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

```
    }

    totalAlunos--;

    cadastro = (Aluno *)realloc(cadastro, totalAlunos * sizeof(Aluno));

    printf("Aluno excluído com sucesso!\n");

    return;

}

}

printf("Aluno com RGM %d não encontrado.\n", rgm);
}

// Menu principal

int main() {

    int opcao;

    do {

        printf("\n--- Menu Principal ---\n");

        printf("1. Cadastrar Aluno\n");

        printf("2. Listar Alunos\n");

        printf("3. Atualizar Aluno\n");

        printf("4. Excluir Aluno\n");

        printf("5. Sair\n");

        printf("Escolha uma opção: ");

        scanf("%d", &opcao);

        switch (opcao) {

            case 1: createAluno(); break;

            case 2: readAlunos(); break;

            case 3: updateAluno(); break;
```

Técnicas de Desenvolvimento de Algoritmos
Prof. Wallace Bonfim

```
case 4: deleteAluno(); break;

case 5: printf("Encerrando o programa.\n"); break;

default: printf("Opção inválida.\n");

}

} while (opcao != 5);


// Libera a memória antes de encerrar o programa
for (int i = 0; i < totalAlunos; i++) {

    free(cadastro[i].nome);

}

free(cadastro);


return 0;

}
```